UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Departamento de Ingeniería Sistemas y Automática

DISEÑO Y REALIZACIÓN DE UNA PLATAFORMA MÓVIL PARA INTERACCIÓN CON IPHONE

PROYECTO FIN DE CARRERA

AUTOR: ALEJANDRO JOSÉ IBÁÑEZ DE LA PUENTE

DIRECTOR: CONCEPCIÓN ALICIA MONJE MICHARET

Jueves 23 de octubre de 2014

PROYECTO FIN DE CARRERA

DISEÑO Y REALIZACIÓN DE UNA PLATAFORMA MÓVIL PARA INTERACCIÓN CON IPHONE

Por

Alejandro José Ibáñez de la Puente

Presentado en la

ESCUELA POLITÉCNICA SUPERIOR

de la

UNIVERSIDAD CARLOS III DE MADRID

Para la obtención del título de

INGENIERO INDUSTRIAL

Directora de Proyecto Fin de Carrera

Dña. Concepción Alicia Monje Micharet

Madrid, 23 Octubre 2014

PROYECTO FIN DE CARRERA

DISEÑO Y REALIZACIÓN DE UNA PLATAFORMA MÓVIL PARA INTERACCIÓN CON IPHONE

Por

Alejandro José Ibáñez de la Puente

Directora de Proyecto Fin de Carrera

Dra. Concepción Alicia Monje Micharet

TRIBUNAL CALIFICADOR

Presidente

María Dolores Blanco Rojas

Secretario

Fernando Martín Monar

Vocal

Cristina Castejón Sisamón

Madrid, 23 de Octubre de 2014

PROYECTO FIN DE CARRERA

ÍNDICE

ÍNDICE **4**

ÍNDICE DE FIGURAS **7**

AGRADECIMIENTOS **8**

RESUMEN **8**

ABSTRACT **9**

1 INTRODUCCIÓN **10**
1.1 Introducción **10**
1.2 Objetivos **10**
1.2.1 Requisitos de la interfaz 11
1.2.2 Requisitos de la plataforma 12
1.3 Estructura de la memoria **13**
1.4 Diagrama de Gantt **15**

2 PROTOCOLO DE COMUNICACIÓN BLUETOOTH **16**
2.1 Bluetooth 4.0 o BLE (Low Energy Bluetooth) **16**
2.2 Librería Bluetooth para Xcode **16**
2.2.1 Detección de un periférico - findBLEPeripherals 17
2.2.2 Conexión a un periférico - connectPeripheral 18
2.2.3 Búsqueda de servicios y características 19
2.2.4 Solicitud de notificación del valor de una característica 20
2.2.5 Actualización del valor de una característica 21
2.2.6 Cambiar el valor de la característica - write 22
2.2.7 Anexo 23
2.3 Librería Bluetooth para Arduino **23**
2.3.1 void BLEMini_begin(unsigned long bound); 24
2.3.2 int BLEMini_available(); 24
2.3.3 int BLEMini_read(); 24
2.3.4 void BLEMini_write(unsigned char dat); 24

3 DISEÑO DE LA INTERFAZ GRÁFICA EN XCODE **25**
3.1 Xcode **25**
3.1.1 Introducción 25
3.1.2 Lenguaje dinámico 25
3.1.3 Clases 26
3.1.4 Objetos 28
3.1.5 Métodos 28
3.1.6 Depuración de errores 30
3.1.7 Intarface Builder 30
3.2 Funcionamiento de la interfaz **31**
3.2.1 Ayuda 32
3.2.2 Búsqueda 32

3.2.3 Conexión ... 33
3.2.4 Joystick ... 35
3.2.5 Acelerómetro ... 35
3.2.6 Actuadores – Sliders y switches ... 36
3.2.7 Web ... 37
3.2.8 Batería baja ... 37
3.2.9 Diagrama de flujo ... 38
3.3 Controles ... 39
3.3.1 Botones ... 39
3.3.2 Sliders ... 41
3.3.3 Segmented Controller ... 41
3.3.4 Interruptores ... 42
3.3.5 Joystick ... 42
3.3.6 Acelerómetro ... 42
4 PROGRAMACIÓN DE LA INTERFAZ ... 43
4.1 Búsqueda de periféricos cercanos ... 43
4.2 Conexión con un periférico ... 47
4.3 Comunicación con un periférico ... 48
4.3.1 Enviar dato – Joystick ... 48
4.3.2 Enviar dato – Acelerómetro ... 49
4.3.3 Enviar dato – Actuadores ... 51
4.3.4 Recibir dato ... 51
4.4 Desconexión ... 52
4.4.1 Desconexión – Botón/timeout ... 52
4.4.2 Desconexión – Cierre de la aplicación ... 53
5 DISEÑO DE LA PLATAFORMA ... 55
5.1 Introducción ... 55
5.2 PCB ... 55
5.2.1 Introducción ... 55
5.2.2 Regulador de tensión (5v) – Electrónica ... 56
5.2.3 Regulador de tensión (7v) – Motores ... 56
5.2.4 Controlador de los motores – Puente H ... 57
5.2.5 Diseño del circuito impreso ... 58
5.3 Prototipado ... 60
5.3.1 Introducción ... 60
5.3.2 Funcionamiento ... 60
5.3.3 Diseño de las piezas ... 60
5.4 Ruedas motrices ... 62
5.4.1 Motor ... 62
5.4.2 Reductor ... 62
5.4.3 Rueda ... 63
5.5 Actuadores ... 63
5.5.1 Servo ... 63
5.5.2 Led ... 64
5.6 Sensores ... 64
5.6.1 Ultrasonidos ... 64
5.7 Ensamblaje final ... 65

6 PROGRAMACIÓN DE LA PLATAFORMA 67
6.1 Arduino 67
6.1.1 Introducción 67
6.1.2 Especificaciones 67
6.1.3 Programación 69
6.1.4 Diagrama de flujo 69
6.2 BLE Mini 71
6.2.1 Características 71
6.2.2 Diseño 72
7 DESPACHO ECONÓMICO 74
7.1 Presupuesto 74
7.2 Trabajos futuros 75
8 CONCLUSIONES 76
8.1 Conclusiones sobre el proyecto 76
8.2 Conclusiones personales 76
9 TRABAJOS FUTUROS 78
10 REFERENCIAS 79
11 ANEXOS 81

ÍNDICE DE FIGURAS

Fig. 1.1 Imagen global del proyecto 10
Fig. 1.2 Diagrama de Gantt 15
Fig. 3.1 Interface Builder 31
Fig. 3.2 Interfaz - Ayuda 32
Fig. 3.3 Interfaz - Búsqueda 32
Fig. 3.4 Interfaz - Buscando 33
Fig. 3.5 Interfaz - Conexión 34
Fig. 3.6 Interfaz - Conectado 34
Fig. 3.7 Interfaz - Joystick 35
Fig. 3.8 Interfaz - Acelerómetro 35
Fig. 3.9 Interfaz - Actuadores 36
Fig. 3.10 Interfaz - Web 37
Fig. 3.11 Interfaz - Batería baja 37
Fig. 3.12 Diagrama de flujo Xcode - Main 38
Fig. 3.13 Diagrama de flujo Xcode - Nuevo dato 39
Fig. 3.14 Interface Builder - Nuevo UIButton 40
Fig. 3.15 Interface Builder - Conexión UIButton con implementación 40
Fig. 3.16 Interface Builder - Conexión UIButton con interfaz 41
Fig. 5.1 Electrónica - Puente H 58
Fig. 5.2 Electrónica - PCB 1.0 59
Fig. 5.3 Electrónica - PCB 2.0 59
Fig. 5.4 Plataforma - Chasis 61
Fig. 5.5 Plataforma - Carcasa 61
Fig. 5.6 Plataforma - Tapa de la batería 62
Fig. 5.7 Electrónica - Servo 64
Fig. 5.8 Electrónica - Ultrasonidos 65
Fig. 5.9 Plataforma - Foto 1 65
Fig. 5.10 Plataforma - Foto 2 66
Fig. 5.11 Escultura de Marina Anaya 66
Fig. 6.1 Diagrama de flujo Arduino - Main 70
Fig. 6.2 BLE Mini - Características 71
Fig. 6.3 BLE Mini - Diseño 72
Fig. 7.1 Presupuesto - Plataforma 75
Fig. 7.2 Presupuesto - Actividades de soporte 75
Fig. 7.3 Presupuesto - Trabajos futuros 75

AGRADECIMIENTOS

Quiero agradecer a mi tutora Concepción Alicia Monje la gran oportunidad que ha supuesto para mi este proyecto ya que me ha permitido poner en práctica los conocimientos adquiridos durante estos años de carrera y ampliarlos. He aprendido nuevos lenguajes de programación, nuevos entornos de desarrollo y nuevos programas que de otra manera probablemente no hubiese llegado a aprender.

También quiero darle las gracias a Marina Anaya por hacer este proyecto posible, ya que su trabajo le da sentido a la plataforma.

RESUMEN

Este proyecto surge de la colaboración del Departamento de Señales y Sistemas de la Universidad Carlos III de Madrid con la artista Marina Anaya para realizar una plataforma móvil controlada mediante un iPhone. El objetivo inicial era el de crear un protocolo de comunicación bluetooth entre un microcontrolador Arduino y un dispositivo iOS iPhone, y fue la escultura de Marina lo que le dio forma al proyecto.

El objetivo del proyecto consiste en realizar una plataforma sobre la que repose la escultura realizada por la artista Marina Anaya, que pueda ser controlada remotamente mediante una aplicación para iPhone. De esta forma se consigue crear una escultura que puede estar ubicada en cualquier punto de la ciudad, y un usuario anónimo la puede controlar descargándose la aplicación de la AppStore de Apple. Para conseguir el objetivo fijado, se divide el proyecto en las tres partes que se listan a continuación.

El protocolo de comunicación entre ambos componentes del proyecto, de manera que se pueda realizar el control de la plataforma desde cualquier dispositivo iPhone que tenga descargada la aplicación eTheatre.

Una interfaz que posee los controles necesarios para que el usuario pueda buscar, conectarse y actuar sobre una plataforma.

Y por último, el diseño y prototipado de una plataforma móvil, tanto desde el punto de vista estructural como del circuito impreso, la electrónica y la programación del procesador. Sobre esta plataforma reposará la escultura diseñada por Marina Anaya.

ABSTRACT

This project is a collaboration between the Department of Signals and Systems of The University Carlos III of Madrid and artist Marina Anaya for a mobile platform controlled by an iPhone. The initial goal was to create a bluetooth communication protocol between an Arduino microcontroller and iPhone iOS device and Marina sculpture was what gave a meaning to the project.

The projects main objective is to make a platform, on which to rest the sculpture made by the artist Marina Anaya, that can be controlled remotely via an iPhone application. By doing this, we created a sculpture that can be located anywhere in the city, and be controlled by an anonymous user by downloading the application from the Apple AppStore. To achieve it, the project is divided into three parts listed below.

The communication protocol between the two components of the project, so any iPhone device that has downloaded the application eTheatre can handle the platform.

An interface that has all the controls needed to allow the user to search, connect and interact with a platform.

And finally, the structural and electronic, design and prototyping of the mobile platform.

On top of this platform will rest sculpture designed by Marina Anaya.

1 INTRODUCCIÓN

1.1 Introducción

La gran importancia de las comunicaciones hoy en día, el creciente uso de los robots en entornos cotidianos y el gran crecimiento que han tenido los smartphones en nuestra sociedad permitió que surgiese este proyecto cuya finalidad era crear una plataforma ubicada en un punto cualquiera de la ciudad de manera que un individuo anónimo pudiese controlarla e interactuar con ella con solo descargarse una aplicación de la AppStore de Apple.

Es un proyecto de gran interés ya que unifica conocimientos de diferentes ramas de la ingeniería para dar lugar a una plataforma que no es exclusivamente funcional, sino que además, gracias al excelente trabajo de la artista Marina Anaya, tiene un gran atractivo visual. Todo esto unido hace que la plataforma sea muy interesante ya que no suele ser habitual un grado de interactuación similar con una obra de arte.

1.2 Objetivos

En el siguiente apartado se desarrollan los objetivos que se establecieron al inicio para su realización durante el proyecto. Se describen los requisitos, tanto para la interfaz como para la plataforma, que fueron definidos conjuntamente entre la directora de proyecto, Concepción Alicia Monje, y la artista, Marina Anaya.

Antes de entrar en detalle en los objetivos del proyecto, se muestra un gráfico orientativo del funcionamiento global del mismo, siendo la línea verde la correspondiente al movimiento, las dos líneas azules corresponden a la comunicación bidireccional entre la plataforma y el dispositivo, y las líneas negras al control de distancia por ultrasonidos. A modo también orientativo, se puede ver la disposición final de la escultura sobre la plataforma móvil.

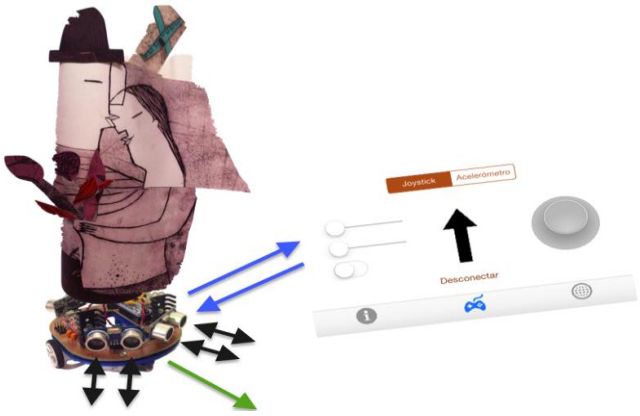


Fig. 1.1 Imagen global del proyecto

1.2.1 Requisitos de la interfaz

Al inicio del desarrollo de la aplicación, se definen una serie de requerimientos que ésta debe cumplir. Estos requerimientos fueron divididos en módulos funcionales que corresponden a las diferentes secciones que componen la App, y que se listan a continuación.

1.2.1.1 Ayuda

La primera pantalla que vemos al arrancar la aplicación debe ser una sencilla guía de utilización de la interfaz de manera que el usuario se familiarice con su funcionamiento.

1.2.1.2 Control de la plataforma

Después, el usuario puede navegar entre las tres pestañas: ayuda, controles y web. La pestaña de los controles tiene las siguientes características.

- Inicialización/búsqueda: Aquí se detectan la/s plataforma/s cercana/s, estableciendo una comunicación inicial con ellas para conocer su disponibilidad y nivel de batería.

- Conexión: Se le muestra al usuario un listado con la/s plataforma/s cercana/s de manera que él pueda decidir qué plataforma quiere controlar.

- Control: Una vez realizada la conexión, se muestran los controles de la plataforma.

    - Joystick: Se selecciona por defecto el control por joystick. Éste consta de una bola redonda a la derecha de la pantalla que el usuario puede mover y manipular alrededor de la pantalla, para controlar la dirección y el sentido de movimiento de la plataforma.

    - Acelerómetro: La interfaz nos permite seleccionar el control de la plataforma mediante la inclinación del dispositivo. Esto se consigue gracias al acelerómetro integrado en el iPhone.

    - Actuadores: Además de controlar la dirección y el sentido del movimiento, el usuario tiene el control de distintos actuadores situados en la estructura: dos servos para poder cambiar el ángulo de las articulaciones, y un led con posibles funciones de encendido, apagado y parpadeo.

- Desconexión: Para dejar la plataforma libre y que pueda ser utilizada por otro usuario hay distintas formas de desconexión.

- Interfaz: A través de la interfaz mediante la pulsación del botón “desconexión”, llevando al usuario otra vez a la pantalla de inicio.

- Límite de tiempo: Si el usuario se mantiene inactivo y no interactúa con la plataforma durante 60 segundos consecutivos, ésta se desconectará automáticamente y enviará la correspondiente señal de control al dispositivo, llevando al usuario otra vez a la pantalla de inicio.

- Aplicación en reposo: Si se sale de la aplicación mientras hay una conexión activa con la plataforma, se envía una mensaje de desconexión a la plataforma. Cuando el usuario vuelva a la aplicación se le mostrará otra vez la pantalla de inicio.

1.2.1.3 Acceso a la web

Se establece la incorporación en la interfaz de una pestaña desde la que se pueda visualizar la web de la artista Marina Anaya, donde aparecerá mas información a cerca del proyecto y el usuario podrá ver el resto de su trabajo.

1.2.1.4 Disponibilidad de la App

Hay una serie de requisitos que, sin intervenir en el desarrollo visual de la interfaz, son básicos para el proyecto ya que en ellos se tienen en cuenta detalles sobre la disponibilidad de la aplicación.

- La aplicación está disponible para dispositivos con iOS, a partir de la versión 7.0.

- La aplicación será especialmente diseñada para dispositivos iPhone e iPod Touch, aunque también podrá ser utilizada en dispositivos iPad.

- Se podrá descargar la aplicación desde la web de Marina Anaya a través de un link, o directamente desde la AppStore de Apple.

1.2.2 Requisitos de la plataforma

El requisito principal es la viabilidad tanto técnica como económica de la plataforma. Al ser un proyecto diseñado para una aplicación real, la plataforma debe poderse llevar a cabo al final del mismo. Las características de diseño de la plataforma de acuerdo con su funcionalidad son: que debe ser compacta, de tamaño reducido, estable y debe tener una superficie diáfana sobre la que pueda reposar la escultura realizada por la artista Marina Anaya.

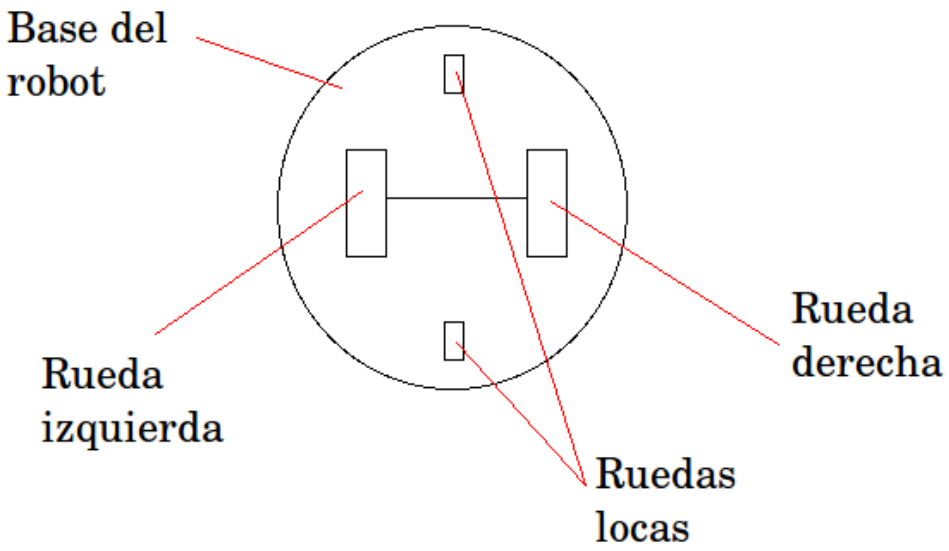
Se definen ahora los requisitos que debe cumplir la plataforma para conseguir implementar el control diseñado en la interfaz. Los dividiremos por funciones en distintos apartados.

1.2.2.1 Control

Se decide utilizar un microcontrolador Arduino Nano junto con un integrado BLE Mini (antena bluetooth) para poder ejecutar el bucle de control, establecer la comunicación, realizar las mediciones con los ultrasonidos, y actuar sobre: el movimiento de la plataforma, el led y los servos.

1.2.2.2 Movimiento

La plataforma debe poder rotar sobre sí misma y moverse en cualquier dirección y sentido. Para esto se propone un robot diferencial con dos ruedas paralelas sobre en el mismo eje de acción y dos rodamientos que aporten estabilidad.



1.2.2.3 Alimentación

Son necesarios dos niveles distintos de tensión: 5v para alimentar la electrónica de la plataforma, y 7v para alimentar los motores.

1.3 Estructura de la memoria

En este apartado se muestra un resumen de todos los capítulos que componen el proyecto fin de carrera.

- *CAPÍTULO 1 "INTRODUCCIÓN":* Se expone una breve introducción acerca del contenido del documento, así como sus objetivos. También muestra la estructura que forma el proyecto fin de carrera.

- *CAPÍTULO 2 "PROTOCOLO DE COMUNICACIÓN BLUETOOTH":* Se incluye una breve introducción al funcionamiento del nuevo protocolo, Bluetooth 4.0 o Low Energy Bluetooth y se explica el código de las librerías realizadas para establecer la comunicación, tanto del iPhone como del Arduino.

- *CAPÍTULO 3 "DISEÑO DE LA INTERFAZ GRÁFICA EN XCODE":* Se explica el criterio que se ha seguido a la hora de mostrar al usuario los controles de la plataforma y cómo se han programado en el Storyboard de Xcode. Además se muestra cómo crea el usuario el enlace entre su iPhone y la plataforma, y cómo se comunica con la misma.

- *CAPÍTULO 4 "PROGRAMACIÓN DE LA INTERFAZ":* Se detalla toda la programación de la interfaz creada, describiendo su estructura general y las funciones usadas.

- *CAPÍTULO 5 "DISEÑO DE LA PLATAFORMA":* En este apartado aparecen explicados los documentos necesarios para la fabricación de la plataforma y el circuito impreso (PCB).

- *CAPÍTULO 6 "PROGRAMACIÓN DE LA PLATAFORMA":* Se explica el código del microcontrolador Arduino Nano.

- *CAPÍTULO 7 "DESPACHO ECONÓMICO":* Se incluye un presupuesto detallado de todos los componentes del proyecto.

- *CAPÍTULO 8 "CONCLUSIONES":* Se hace un análisis sobre los resultados del trabajo.

- *CAPITULO 9 "TRABAJOS FUTUROS":* Se comentan las posibles mejoras del trabajo.

- *CAPÍTULO 10 "REFERENCIAS":* Se muestran las fuentes consultadas a lo largo del proyecto.

- *CAPÍTULO 11 "ANEXOS":* Se incluye el código que compone el proyecto y la documentación de alguno de los componentes utilizados en la placa.

1.4 Diagrama de Gantt

En la planificación inicial del proyecto no se decidió una duración exacta del mismo, pero si que se puso como propósito inicial defender en el curso 13/14, por lo que el proyecto se inició a mediados del primer cuatrimestre para tener tiempo suficiente para entregarlo al final del verano.

Hubo dos periodos en los que el proyecto tuvo un parón, y corresponden a las épocas de examen de la escuela para las convocatorias ordinarias y extraordinaria de los meses de Enero, Mayo y Junio.

El diagrama de Gantt del proyecto queda así.

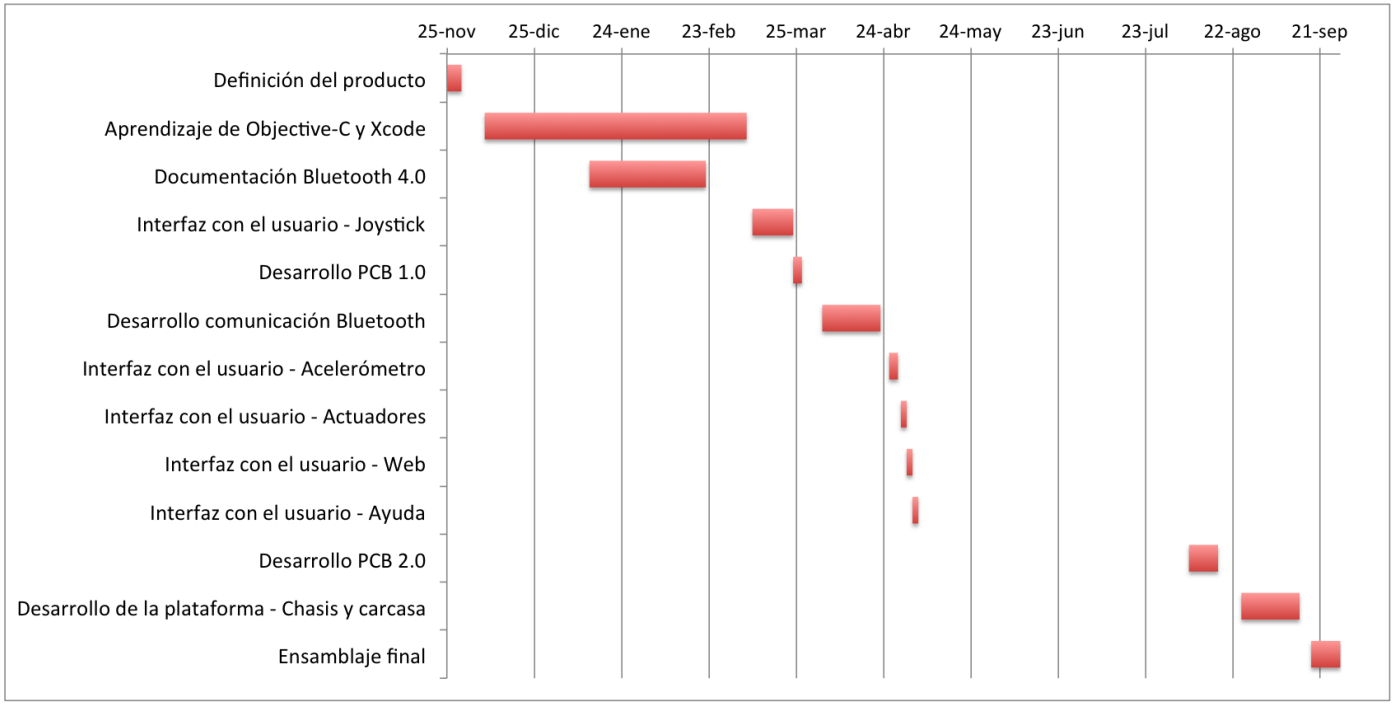


Fig. 1.2 Diagrama de Gantt

2 PROTOCOLO DE COMUNICACIÓN BLUETOOTH

2.1 Bluetooth 4.0 o BLE (Low Energy Bluetooth)

Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz. Los principales objetivos que se pretenden conseguir con esta norma son[4]:

- Facilitar las comunicaciones entre equipos móviles.
- Eliminar los cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

En el protocolo de comunicación Bluetooth Low Energy, los actores principales son las centrales y los periféricos. Un periférico típicamente tiene información que necesitan otros dispositivos y una central se encarga de utilizar la información provista por el periférico para realizar alguna tarea. Todas estas comunicaciones se basan en el protocolo GATT (Generic Attribute Profile). Este protocolo dispone de una serie de comandos que permite al cliente (central) obtener información a cerca del servidor (periférico) como puede ser descubrir su identificador, sus servicios o las características asociadas a cada servicio. A el conjunto de servicios, características y descriptores se les conoce como atributos y cada uno tiene su propio UUID, de manera que si conozco previamente el atributo del cual quiero conocer su valor, sólo he de buscar su identificador. Esto permite al dispositivo suscribirse únicamente el atributo que nos interesa lo cual reduce el consumo de energía[4].

Una vez la conexión ha sido establecida, el periférico retransmitirá periódicamente el descriptor al cual la central esta suscrita hasta que el enlace sea finalizado por la central o se vea interrumpido por algún error en la comunicación.

2.2 Librería Bluetooth para Xcode

Ahora vamos a ver los pasos que he de seguir el iPhone desde que detectamos el dispositivo deseado hasta que nos suscribimos a la característica asociada al servicio que estamos buscando.

Las métodos que aparecen a la derecha de cada título son los que debemos de llamar desde la interfaz para que tenga lugar la acción que queremos realizar. En caso de que no aparezca ningún método esto quiere decir que el código seguirá ejecutándose automáticamente mediante los métodos que vayamos llamando y que se vayan invocando desde el propio gestor central[4].

2.2.1 Detección de un periférico - findBLEPeripherals

Lo primero que hemos de hacer es declarar las propiedades que vamos a utilizar a la hora de realizar la conexión en el .h de mi librería.

```
@property (strong, nonatomic) NSMutableArray *peripherals;
@property (strong, nonatomic) CBCentralManager *CM;
@property (strong, nonatomic) CBPeripheral *activePeripheral;
```

Dado que el `CBCentralManager` es la representación en la programación orientada a objetos de una central Bluetooth, hay que asignarle espacio en memoria e inicializarla antes de poder realizar ninguna acción con ella. Esto se consigue con el siguiente método[3].

```
self.CM = [[CBCentralManager alloc] initWithDelegate:self queue:nil];
```

El siguiente paso sería buscar los periféricos cercanos mediante el método[8].

```
[self.CM scanForPeripheralsWithServices:nil options:nil];
```

Mediante el primer parámetro introducido, podemos buscar un periférico cualquiera nil, o bien uno con un UUID determinado. En este caso vamos a buscar un periférico con un identificador determinado que viene asignado por el fabricante[3] [8].

```
#define RBL_SERVICE_UUID "713D0000-503E-4C75-BA94-3148F18D941E"

- (int) findBLEPeripherals:(int) timeout
{
    if (self.CM.state != CBCentralManagerStatePoweredOn)
    {
        NSLog(@"CoreBluetooth no ha sido inicializado correctamente");
        NSLog(@"Estado = %ld (%s)\r\n", (unsigned long) self.CM.state, [self
centralManagerStateToString:self.CM.state]);
        return -1;
    }

    [NSTimer scheduledTimerWithTimeInterval:(float)timeout target:self
selector:@selector(scanTimer:) userInfo:nil repeats:NO];

    [self.CM scanForPeripheralsWithServices:[NSArray arrayWithObject:[CBUUID
UUIDWithString:@RBL_SERVICE_UUID]] options:nil];

    NSLog(@"Iniciando escaneado de periféricos con el UUID determinado");
    return 0;

}
```

Una vez se ha detectado el periférico que contiene el identificador buscado, el gestor central (central delegate) llama al método didDiscoverPeripheral.

El periférico encontrado es devuelto como un objeto `CBPeripheral`. Ahora debemos almacenarlo en el vector de periféricos definido previamente en el .h para poder conectarnos a el posteriormente.

```
- (void) centralManager : (CBCentralManager *) central didDiscoverPeripheral : (CBPeripheral *)
peripheral advertisementData : (NSDictionary *)advertisementData RSSI:(NSNumber *)RSSI {


  if (!self.peripherals)
    self.peripherals = [[NSMutableArray alloc] initWithObjects:
            peripheral,nil];
  else {
    for(int i = 0; i < self.peripherals.count; i++){
      CBPeripheral *p = [self.peripherals objectAtIndex:i];

      if ((p.identifier == NULL) || (peripheral.identifier == NULL))
       continue;

      if ([self UUIDSAreEqual:p.identifier UUID2:
                    peripheral.identifier ]){
       [self.peripherals replaceObjectAtIndex:i withObject:peripheral];
       NSLog(@"Periférico encontrado previamente");
       return;
     }
    }
    [self.peripherals addObject:peripheral];
    NSLog(@"Nuevo periférico añadido");
  }
}
```

Una vez hemos detectado todos los periféricos deseados, para ahorrar batería y mejorar la velocidad de proceso, dejamos de escanear en busca de mas dispositivos. Esto lo hacemos mediante la función stopScan que teníamos programada para que se ejecutase 1 segundo después de haber iniciado la búsqueda[6].

```
- (void) scanTimer:(NSTimer *)timer
{
   [self.CM stopScan];
   NSLog(@"Dejamos de escanear");
   NSLog(@"número de periféricos conocidos : %lu", (unsigned long)[self.peripherals count]);

}
```

2.2.2 Conexión a un periférico - connectPeripheral

Ahora vamos a ver el proceso para conectarnos a un periférico de la lista de periféricos encontrados anteriormente y que había sido almacenada en el vector peripherals[2].

Para solicitar una conexión con el periférico determinado hemos de hacerlo mediante el método connectPeripheral: Lo invocaremos desde un método propio que hemos llamado de la misma forma, de manera que luego podamos acceder a él desde la interfaz introduciendo en el parámetro peripheral el periférico al que nos queremos conectar[3].

Antes de poder interactuar con el periférico en cuestión, hemos de inicializarlo mediante self.activePeripheral.delegate = self .

```
- (void) connectPeripheral:(CBPeripheral *)peripheral
{
```

```
    NSLog(@"Conectando con el periférico con UUID : %@", peripheral.identifier.UUIDString);

    self.activePeripheral = peripheral;
    self.activePeripheral.delegate = self;
    [self.CM connectPeripheral:self.activePeripheral options:[NSDictionary
dictionaryWithObject:[NSNumber numberWithBool:YES]
forKey:CBConnectPeripheralOptionNotifyOnDisconnectionKey]];

}
```

Una vez nos hemos conectado, se invocará el método didConnectPeripheral. Esto implica que ya estaremos conectados al periférico deseado. Ahora empezaremos a explorar la información del mismo. Lo primero que debemos conocer son los servicios disponibles mediante el método discoverServices. Al introducir el parámetro nil le estamos diciendo que busque todos los servicios disponibles de éste periférico[3].

```
- (void)centralManager:(CBCentralManager *)central didConnectPeripheral: (CBPeripheral
*)peripheral
{
    self.activePeripheral = peripheral;
    [self.activePeripheral discoverServices:nil];

}
```

2.2.3 Búsqueda de servicios y características

Una vez que se han encontrado los servicios especificados, se invocará el método didDiscoverServices. Dentro del mismo se comprueba si se ha recibido un mensaje de error en caso de no haber descubierto ningún servicio, en caso contrario, buscamos todas las características[2] asociadas a cada uno de los servicios descubiertos mediante la función propia getAllCharacteristicsFromPeripheral[3]

```
- (void)peripheral:(CBPeripheral *)peripheral didDiscoverServices: (NSError *)error
{
    if (!error)
    {
        [self getAllCharacteristicsFromPeripheral:peripheral];
    }
    else
    {
        NSLog(@"No se encontró ningún servicio");
        [[self delegate] bleNoPudoConectar];
    }

}
```

En la función getAllCharacteristicsFromPeripheral llamaremos al método discoverCharacteristics de la clase `CBPeripheral` para cada uno de los servicios del periférico.

```
-(void) getAllCharacteristicsFromPeripheral:(CBPeripheral *)p
{
    for (int i=0; i < p.services.count; i++)
    {
        CBService *s = [p.services objectAtIndex:i];
```

```
        [p discoverCharacteristics:nil forService:s];
    }
}
```

2.2.4 Solicitud de notificación del valor de una característica

Una vez encontradas todas las características de cada uno de los servicios, ya podemos solicitar que se nos notifique el valor de la característica deseada mediante el método setNotifyValue. En este caso ya no queremos suscribirnos a todas las características por lo que en vez de introducir el parámetro nil, introduciremos el identificador UUID de la característica correspondiente[3] [9].

```
 - (void)peripheral:(CBPeripheral *)peripheral didDiscoverCharacteristicsForService:(CBService
*)service error:(NSError *)error
{
    if (!error)
    {
        for (int i=0; i < service.characteristics.count; i++)
        {
            CBService *s = [peripheral.services objectAtIndex:
                            (peripheral.services.count - 1)];

            if ([service.UUID isEqual:s.UUID])
            {
                if (!done)
                {
                    [self enableReadNotification:activePeripheral];
                    [[self delegate] bleConectado];
                    isConnected = true;
                    done = true;
                }
                break;
            }
        }
    }
    else
    {
        NSLog(@"No se encontró la característica buscada");
        [[self delegate] bleNoPudoConectar];
    }
}

-(void) enableReadNotification:(CBPeripheral *)p
{
    CBUUID *uuid_service = [CBUUID UUIDWithString:@RBL_SERVICE_UUID];
    CBUUID *uuid_char = [CBUUID UUIDWithString:@RBL_CHAR_TX_UUID];

    [self notification:uuid_service characteristicUUID:uuid_char p:p
          on:YES];

}

-(void) notification:(CBUUID *)serviceUUID characteristicUUID:(CBUUID *)characteristicUUID
p:(CBPeripheral *)p on:(BOOL)on
{
    NSLog(@"Habilitando notificaciones");
    CBService *service = [self findServiceFromUUID:serviceUUID p:p];
```

```
    if (!service)
    {
        NSLog(@"No se pudo encontrar el servicio con UUID %@, del periférico con UUID %@", [self
CBUUIDToString:serviceUUID], p.identifier.UUIDString);
        [[self delegate] bleNoPudoConectar];
        return;
    }

    CBCharacteristic *characteristic = [self findCharacteristicFromUUID:characteristicUUID
service:service];

    if (!characteristic)
    {
        NSLog(@"No se pudo encontrar la característica con UUID %@, en el servicio UUID %@, del
periférico UUID %@", [self CBUUIDToString: characteristicUUID], [self
CBUUIDToString:serviceUUID], p.identifier.UUIDString);
        [[self delegate] bleNoPudoConectar];
        return;
    }

    [p setNotifyValue:on forCharacteristic:characteristic];

}
```

2.2.5 Actualización del valor de una característica

Finalmente después de suscribirnos a la característica, cada vez que esta cambie su valor se ejecutará el método didUpdateValueForCharacteristic. Ahora nosotros ya solo tenemos que procesar la información recibida de manera que sea comprensible para nuestra interfaz.

```
- (void)peripheral:(CBPeripheral *)peripheral didUpdateValueForCharacteristic:(CBCharacteristic
*)characteristic error:(NSError *)error
{
    NSLog(@"Actualizando valor de la característica");
    unsigned char data[20];
    static unsigned char buf[512];
    static int len = 0;
    NSInteger data_len;

    if (!error)
    {
        if ([characteristic.UUID isEqual:[CBUUID UUIDWithString:
                @RBL_CHAR_TX_UUID]])
        {
            data_len = characteristic.value.length;
            [characteristic.value getBytes:data length:data_len];

            if (data_len == 20)
            {
                memcpy(&buf[len], data, 20);
                len += data_len;

                if (len >= 64)
                {
                    [[self delegate] bleNuevoDato:buf length:len];
                    len = 0;
                }
            }
            else if (data_len < 20)
```

```
            {
                memcpy(&buf[len], data, data_len);
                len += data_len;

                [[self delegate] bleNuevoDato:buf length:len];
                len = 0;
            }
        }
    }
    else
    {
        NSLog(@"Error al actualizar el valor de la característica");
    }
}
```

Una vez que se ha finalizado este proceso ya sólo nos quedaría controlar desde la interfaz el valor de la característica a través del método que hemos llamado bleNuevoDato, que se invocará cada vez que ésta de actualice.

2.2.6 Cambiar el valor de la característica - write

El servicio de la antena Bluetooth que hemos comprado nos permite acceder a dos características distintas, una para recibir y otra para enviar información, de manera que la comunicación con el dispositivo sea bidireccional. Mediante el método write, podemos enviar un vector de bytes sin signo al dispositivo[3].

```
-(void) write:(NSData *)d
{
    CBUUID *uuid_service = [CBUUID UUIDWithString:@RBL_SERVICE_UUID];
    CBUUID *uuid_char = [CBUUID UUIDWithString:@RBL_CHAR_RX_UUID];

    [self writeValue:uuid_service characteristicUUID:uuid_char p:activePeripheral data:d];
}

-(void) writeValue:(CBUUID *)serviceUUID characteristicUUID:(CBUUID *)characteristicUUID
p:(CBPeripheral *)p data:(NSData *)data
{
    CBService *service = [self findServiceFromUUID:serviceUUID p:p];

    if (!service)
    {
        NSLog(@"No se pudo encontrar el servicio con UUID %@, del periférico con UUID %@", [self
CBUUIDToString:serviceUUID], p.identifier.UUIDString);

        return;
    }

    CBCharacteristic *characteristic = [self findCharacteristicFromUUID:characteristicUUID
service:service];

    if (!characteristic)
    {
        NSLog(@"No se pudo encontrar la característica con UUID %@, en el servicio UUID %@, del
periférico UUID %@", [self CBUUIDToString:characteristicUUID], [self
CBUUIDToString:serviceUUID], p.identifier.UUIDString);

        return;
```

```
    }

    [p writeValue:data forCharacteristic:characteristic type:CBCharacteristicWriteWithoutResponse];
}
```

Es la función `[p writeValue:data forCharacteristic:characteristic type:CBCharacteristicWriteWithoutResponse];` la que finalmente actualiza el valor de nuestra característica haciéndosela llegar al dispositivo móvil[3].

2.2.7 Anexo

Aquí se incluyen dos métodos que han sido utilizados en varios puntos del código a la hora de comprobar en que índice de nuestro vector de servicios estaba el deseado `findServiceFromUUID`. De la misma forma ocurre con el método `findCharacteristicFromUUID` para encontrar la característica deseada dentro del conjunto de las mismas asociadas al servicio determinado anteriormente.

```
-(CBService *) findServiceFromUUID:(CBUUID *)UUID p:(CBPeripheral *)p
{
    for(int i = 0; i < p.services.count; i++)
    {
        CBService *s = [p.services objectAtIndex:i];
        if ([self compareCBUUID:s.UUID UUID2:UUID])
            return s;
    }

    return nil; //Servicio no encontrado en este periférico
}

-(CBCharacteristic *) findCharacteristicFromUUID:(CBUUID *)UUID service:(CBService*)service
{
    for(int i=0; i < service.characteristics.count; i++)
    {
        CBCharacteristic *c = [service.characteristics objectAtIndex:i];
        if ([self compareCBUUID:c.UUID UUID2:UUID])
            return c;
    }

    return nil; // Característica no encontrada en este servicio

}
```

2.3 Librería Bluetooth para Arduino

La realización de esta parte del PFC dependía de la elección de la antena bluetooth que fuese a ser utilizada ya que los propios fabricantes tienen sus librerías para establecer un protocolo de comunicación serie con el Arduino gracias a la UART del procesador, de manera que pudiésemos leer y escribir las características bluetooth asociadas a la antena[11].

En este caso, se eligió una antena de bajo coste y tamaño reducido del fabricante RedBearLab. De los múltiples productos de este fabricante se eligió el BLE Mini, una antena basada en el integrado de Texas Instruments CC2540 que le

aporta una gran versatilidad al proyecto al tener un tamaño muy reducido (L)39mm x (W)18.5mm x (H)3.8mm, y le dota de una gran sencillez al tener un puerto de 4 pines exclusivamente (VCC,GND,TX y RX), lo cual la convierte en la mejor opción dadas las necesidades del proyecto[12].

Esta antena establece un protocolo de comunicación serie con el Arduino. Las funciones de la librería utilizada para establecer y controlar dicho protocolo son privadas y no podemos acceder al código, pero si que se expondrán aquí y se explicará la lógica asociada a ellas.

2.3.1 void BLEMini_begin(unsigned long bound);

Esta función se encarga de inicializar el protocolo de comunicación serie entre el Arduino y la antena. A la variable bound se le asigna un valor en baudios de 57600 ya que es la velocidad a la que viene pre-programada la antena[12].

2.3.2 int BLEMini_available();

Esta función nos devuelve el número de bytes recibidos en la última actualización de la característica RX de la antena. En nuestra librería en Xcode hemos codificado nuestro protocolo de comunicación en series de 3 bytes sin signo, de manera que cuando esta función nos devuelva el número entero 3, sabremos que hemos recibido un nuevo dato completo[12].

2.3.3 int BLEMini_read();

Esta función nos devuelve byte a byte el valor de la característica por lo que hemos de ejecutarla 3 veces seguidas para obtener el valor de los 3 bytes donde se encuentra almacenada la información enviada desde el iPhone. Hemos de esperar a leerla completamente antes de que el iPhone (funcionando como una Central Bluetooth) vuelva a actualizar el valor[12].

2.3.4 void BLEMini_write(unsigned char dat);

Esta función se encarga de actualizar el valor de nuestra característica TX de manera que haya realimentación en la comunicación y podamos habilitar al iPhone para enviar un nuevo dato[12].

3 DISEÑO DE LA INTERFAZ GRÁFICA EN XCODE

3.1 Xcode

Xcode es el IDE (Entorno de Desarrollo Integrado) utilizado en el iPhone SDK (Kit de desarrollo de software para iOS) de Apple y se suministra gratuitamente para Mac OSX. Xcode trabaja conjuntamente con Interface Builder, herramienta gráfica para la creación de interfaces de usuario. Incluye la colección de compiladores del proyecto GNU (GCC) para diferentes leguajes, entre ellos Objective-C, el utilizado para el desarrollo de aplicaciones para sistemas operativos iOS[1].

3.1.1 Introducción

Objective-C es un lenguaje de programación orientado a objetos, nacido en la década de los 80, creado como un superconjunto de C. Esto quiere decir que es posible compilar cualquier programa escrito en C con un compilador de Objective-C, y también se puede incluir libremente código en C dentro de una clase de Objective-C.

La interfaz e implementación de una clase se encuentran en bloques de código separados. Al igual que ocurre en C++, las clases de objetos en Objective-C, por lo general se definen mediante un fichero de cabecera (extensión .h) y un fichero de implementación (extensión .m).

Cocoa Touch es un framework para la creación de aplicaciones. Proporciona una capa de abstracción al sistema operativo iOS que incluye reconocimiento gestual, animaciones y una librería distinta para la interfaz de usuario. Junto a InterfaceBuilder permiten generar la interfaz de usuario de la App, almacenando el resultado en un fichero.xib. Las herramientas para desarrollar aplicaciones basadas en Cocoa Touch se incluyen en el SDK de iOS[1].

3.1.2 Lenguaje dinámico

Si hay una característica que diferencia a Objective-C de otros lenguajes, esa sería que es un lenguaje muy dinámico, en el sentido en que muchas decisiones que son tomadas en tiempo de compilación por muchos lenguajes, Objective-C las toma en tiempo de ejecución.

Esta ventaja se aprecia principalmente en las herramientas de desarrollo, ya que estas tienen acceso a todo el runtime del programa, de manera que las herramientas de desarrollo pueden instanciar los objetos del programa, representarlos visualmente, personalizarlos, monitorizarlos, y depurarlos de forma muy cómoda para el programador[1].

3.1.3 Clases

Las clases en Objective-C constan de una interfaz y una implementación. La **interfaz** indica la estructura del objeto, y la **implementación** contiene la implementación de sus métodos.

La interfaz y la implementación se suelen poner en dos ficheros distintos (con las extensiones `.h` y `.m` respectivamente). El código fuente del fichero de interfaz debe estar disponible para ser usado por otros programadores. Sin embargo el fichero de implementación puede entregarse compilado.

Al igual que ocurre en el lenguaje C++, un fichero de implementación puede contener tanto la interfaz como la implementación de la clase, o bien un mismo fichero de implementación puede contener varias implementaciones de clases. En general lo recomendable es poner cada interfaz y cada implementación en un fichero distinto, pero a veces puede resultar útil poner la interfaz y la implementación en un mismo fichero de implementación. Esto ocurre cuando creamos una pequeña clase auxiliar que no vamos a usar más que desde de la implementación de otra clase. En este caso se puede poner la interfaz e implementación de la clase auxiliar dentro del fichero de implementación de la otra clase[1].

- **La Interfaz**

El siguiente fragmento de código muestra un ejemplo de interfaz. Normalmente una clase comienza importando las clases a las que hace referencia (en este caso `NSObject`). La declaración de la interfaz va desde la directiva del compilador `@interface` hasta la directiva del compilador `@end`.

Lo primero que se indica es el nombre de la clase y la clase de la que deriva. En Objective-C si una clase no deriva de `NSObject` (o de `Object`), no puede utilizar las ventajas del runtime de Objective-C y no se podrán ejecutar métodos sobre ella.

En la declaración de la interfaz de una clase, lo primero que aparece son las variables de instancia de ésta. Estas se ponen siempre entre llaves, y su sintaxis es similar a la de otros lenguajes.

Después de cerrar las llaves aparece la declaración de los métodos de la clase. Los métodos que empiezan por `-` son métodos de instancia, y los que empiezan por `+` son métodos de clase. El lenguaje nos obliga a indicar si un método es de instancia o de clase. Las variables de instancia, métodos de instancia, y métodos de clase pueden tener el mismo nombre sin que se produzcan conflictos entre ellos. Por ejemplo, en el código las variables de instancia `x`, `y` tienen el mismo nombre que los métodos `x`, `y`[1].

```
//Punto.h
#import <Foundation/Foundation.h>

@interface Punto : NSObject {
   NSInteger x;
   NSInteger y;
}

- init;
- (NSInteger)x;
- (NSInteger)y;
- (void)setX:(NSInteger)paramX;
- (void)setX:(NSInteger)paramX
@end
```

- **La Implementación**

El siguiente fragmento de código muestra un ejemplo de implementación de la interfaz anterior. Lo primero que se suele hacer en la implementación de una clase es importar su interfaz, y después se usan las directivas del compilador `@implementation` y `@end` para encerrar la implementación de la clase.

```
// Punto.m
#import "Punto.h"
static NSInteger nPuntos=0;

@implementation Punto
- init {
   if (self = [super init]) {
      nPuntos++;
   }
   return self;
}
- (NSInteger)x {
   return x;
}
- (NSInteger)y {
   return y;
}
- (void)setX:(NSInteger)paramX {
   x = paramX;
}
- (void)setY:(NSInteger)paramY {
   y = paramY;
}
```

Dentro de la implementación de una clase no se pueden declarar nuevas variables de instancia, pero sí que es posible declarar métodos en la implementación de una clase que no aparezcan en la interfaz. En este caso los métodos son tratados como privados, y sólo podrán ser llamados desde la implementación de la clase[1].

3.1.4 Objetos

En C++ los objetos se pueden crear tanto en la pila como en memoria dinámica. Por el contrario en Objective-C, los objetos sólo se pueden crear en memoria dinámica. Esto da lugar a que en Objective-C sólo nos podamos referir a los objetos mediante punteros, es decir:

```
Punto p;  // Error de compilación
Punto* p; // correcto
```

También, a diferencia de C++, en Objective-C no se pueden pasar a las funciones objetos por valor, sólo se pueden pasar (y sólo pueden devolver) por referencia, es decir, pasar (y devolver) punteros a objetos.

En Objective-C cuando decimos que `p` es un objeto de la clase `Punto`, lo que debemos entender es que `p` es un puntero a un objeto de la clase `Punto`[1].

- **Instanciar objetos**

El método de clase `alloc` reserva la memoria dinámica del objeto, y pone a cero todas las variables de instancia del objeto. El método de instancia `init` se encarga de inicializar las variables de instancia del objeto.

```
Punto* p = [Punto alloc];
p = [p init];
```

Ambos pasos se pueden realizar en una sola línea de la forma:

```
Punto* p = [[Punto alloc] init];
```

3.1.5 Métodos

Los métodos son operaciones asociadas con un objeto, y se usan, o bien como interfaces para leer y cambiar el estado de un objeto, o bien como un mecanismo para pedir al objeto que realice una acción[1].

- **Declaración de un método**

Las principales partes de la declaración de un método son:

- El **nivel** del método, que es obligatorio, e indica quién recibe el mensaje, si la clase (en cuyo caso se pone un +), o las instancias de la clase (en cuyo caso se pone un −).

- El **tipo de retorno**, que indica el tipo de la variable que retorna el método. En caso de que no retorne nada se usa `void`. El tipo de retorno es opcional, y si no se indica el tipo de retorno, se retorna por defecto el `id`. En los métodos Objective-C es obligatorio poner entre paréntesis los tipos de retorno y de los parámetros.

- El **nombre del método** que es obligatorio y, junto con el nombre de los parámetros, permite identificar de forma única a los métodos de un objeto.

- Los **parámetros** del objeto, los cuales usan una notación infija heredada de Smalltalk. Un método tiene tantos parámetros como veces pongamos el símbolo **:**

- **Implementación de un método**

El cuerpo de un método aparece en el fichero de implementación de la clase. El método empieza con un prototipo idéntico al de la interfaz, excepto que no acaba en punto y coma, sino que entre llaves se indica su implementación. Por ejemplo:

```
- (void)setX:(NSInteger)paramX {
   x = paramX;
}
```

A diferencia de lo que ocurría con las variables de instancia, la implementación de una clase puede tener **métodos privados** que son métodos que no aparecen en la interfaz de la clase.

- **Ejecutar un método**

Siempre que ejecutamos un método tiene que existir un receptor (que puede ser la clase o una instancia de la clase), y un nombre de método usado para indicar el mensaje a enviar. Para ejecutar el método encerramos el nombre del objeto receptor y el del método a ejecutar entre corchetes de la forma:

```
[p setX:5];
```

Una llamada a método es una expresión, con lo que si el método retorna un valor, podemos asignárselo a una variable:

```
NSInteger a = [p x];
```

Las llamadas a métodos también se pueden anidar:

```
[p1 setX:[p x]];
```

- **Encapsulación**

Para poder ocultar las partes del objeto que otros programadores no necesitan conocer para manejar nuestro objeto, Objective-C permite limitar el ámbito desde el que podemos acceder a las variables de instancia de un objeto.

Para declarar los niveles de encapsulación de las variables de instancia se usan los **modificadores de acceso** `@public`, `@protected` y `@private`. Estas directivas del compilador pueden aparecer tantas veces como sea necesario, y afectan a todas las variables de instancia desde su aparición hasta el nuevo modificador de acceso. Si no existe modificador de acceso, por defecto las variables de instancia son `@protected`. Los efectos de los modificadores de acceso son los siguientes:

1. Cuando una variable de instancia tiene el modificador de acceso `@public`, la variable de instancia es accesible desde cualquier parte del programa.

2. Cuando una variable de instancia tiene el modificador de acceso `@private`, entonces si se accede a la variable de instancia desde dentro del objeto la variable de instancia es visible, en cualquier otro caso la variable de instancia no es visible.

3. Cuando una variable de instancia tiene el modificador de acceso `@protected`, las reglas de acceso son similares a las de `@private` excepto que también se permite acceder a la variable de instancia desde una clase derivada.

3.1.6 Depuración de errores

Se puede introducir en distintos puntos dentro del código la función `NSLog(@"...");`. Ésta nos permite depurar el programa, detectando los métodos que se van ejecutando y comprobando los valores de las variables y características durante la ejecución del programa[7].

3.1.7 Intarface Builder

Parte del trabajo de hacer un programa orientado a objetos consiste en definir los diagramas de objetos que muestran las relaciones entre los objetos, y cómo estas relaciones evolucionan a lo largo del ciclo de vida del programa. Algunas relaciones pueden ser totalmente transitorias, mientras que otras se mantienen durante toda la ejecución del programa.

En programación orientada a objetos se suele llamar **asociación** a cualquier tipo de relación entre objetos. Estas asociaciones pueden ser peer-to-peer, es decir donde ningún objeto domina sobre el otro, en cuyo caso se llaman asociaciones **extrínsecas**, o por el contrario ser asociaciones **intrínsecas**, es decir donde un objeto domina sobre otro.

Dos tipo de asociaciones intrínsecas muy conocidas son la composición y la agregación: En la **composición** la vida de un objeto está limitada por la de su contendedor. En la **agregación** la vida del objeto contenido no está estrictamente limitada por la de su contenedor[1].

Interface Builder permite al programador crear un **grafo de objetos** gráficamente y después almacenar este grafo de objetos en un fichero `.nib` (NeXTSTEP Interface Builder). Cuando el usuario ejecute el programa, el grafo de objetos se recuperará del fichero `.nib` a memoria. El fichero `.nib` no sólo almacena los objetos, sino también las conexiones entre los objetos. Se llama **conexión** a cualquier asociación del grafo de objetos que se almacena de forma persistente. Las conexiones se dividen en:

1. **Conexiones outlet**, que son punteros de un objeto a otro objeto.

2. **Conexiones target-action**, las cuales crean relaciones que permiten enviar un mensaje llamado **action** a un objeto llamado **target**.

3. **Bindings**, que permiten mantener sincronizadas las propiedades de dos objetos distintos. Se usan principalmente en Cocoa Bindings con el fin de mantener sincronizadas las propiedades del objeto modelo y objeto vista del patrón de diseño Modelo-Vista-Controlador.

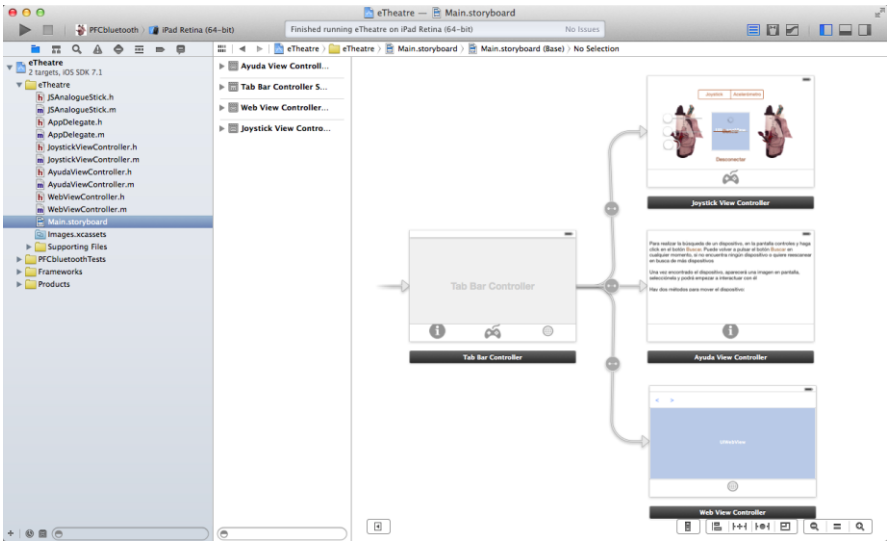


Fig. 3.1 Interface Builder

3.2 Funcionamiento de la interfaz

La interfaz, programada mediante Interface Builder, nos permite controlar la plataforma así como proveernos de un documento de ayuda y de un acceso web a la pagina de Marina Anaya. Se ha diseñado para un dispositivo iPhone, pero gracias a la función Autolayout podemos exportar la App a otros dispositivos iOS con pantallas de tamaño distinto de para el que se diseñó[10].

A continuación se va a exponer el procedimiento necesario para conectarnos con la plataforma apoyado de capturas de imagen que expliquen como se ha realizado la programación desde un punto de vista exclusivamente gráfico. Más

adelante se mostrará y explicará el código asociado a cada uno de los objetos gráficos que vamos a ver ahora.

3.2.1 Ayuda

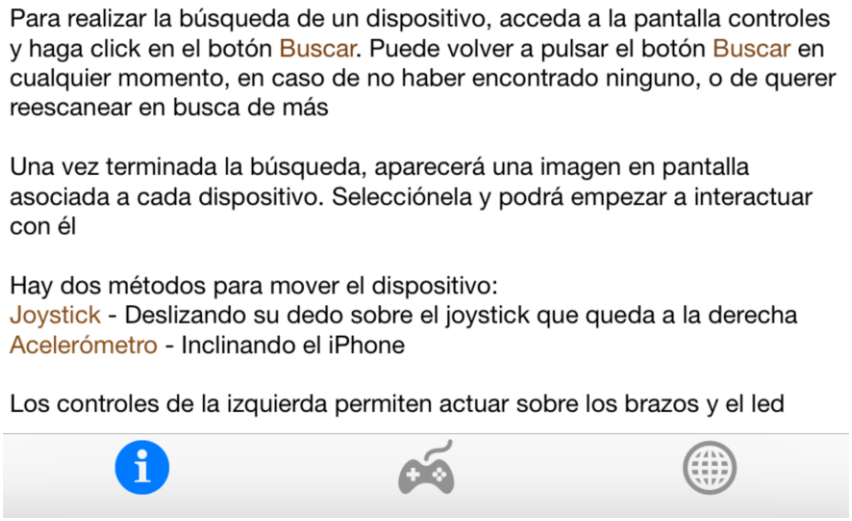


Fig. 3.2 Interfaz - Ayuda

Se visualiza un breve documento, accesible en cualquier momento de la ejecución de la aplicación, que sirva de guía al usuario para que sea capaz de buscar y conectarse a una de las plataformas.

3.2.2 Búsqueda

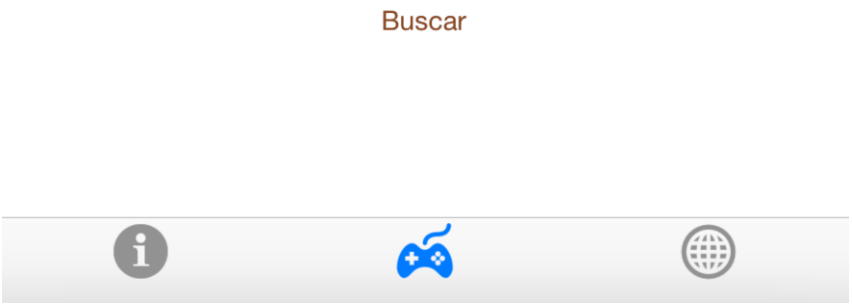


Fig. 3.3 Interfaz - Búsqueda

Una vez en la pestaña desde la que controlaremos la plataforma, lo primero que veremos es el botón de búsqueda.

Acto seguido después de pulsar el botón, se mostrarán un indicador de actividad y una barra de progreso que se irá rellenando en función de los periféricos que vayamos encontrando.

El iPhone comenzará a buscar periféricos cercanos, se conectará a ellos, detectará su disponibilidad, y se desconectará, de manera que después de este proceso podamos mostrar por pantalla al usuario las plataformas que están disponibles.

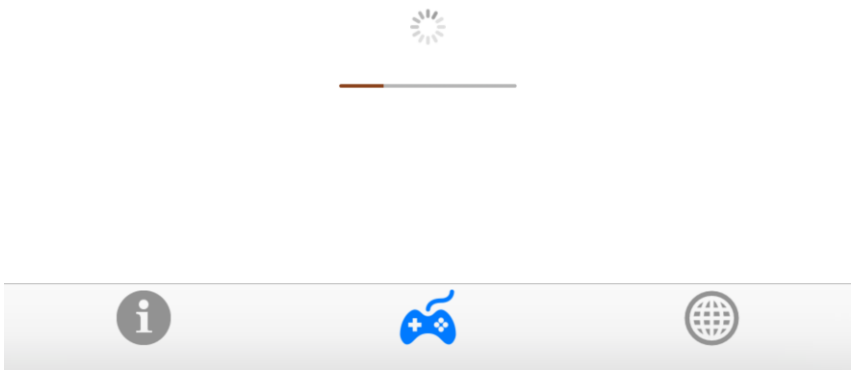


Fig. 3.4 Interfaz - Buscando

3.2.3 Conexión

Una vez terminada la rutina de búsqueda de periféricos, se procederá a mostrar su resultado por pantalla. El botón buscar aparecerá nuevamente independientemente del número de plataformas que hayamos encontrado de manera que se nos permita re escanear en cualquier momento.

El botón que tenemos que pulsar para conectarnos a la plataforma deseada vendrá representado por una imagen de la escultura correspondiente a la plataforma.

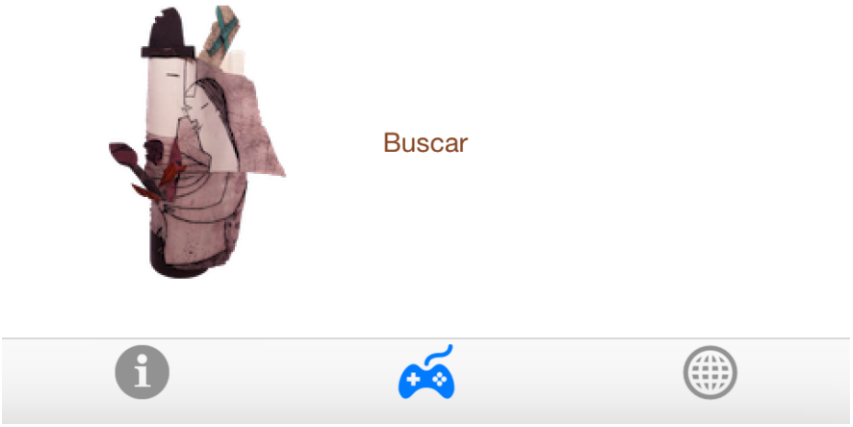


Fig. 3.5 Interfaz - Conexión

Una vez seleccionada una de las plataformas, se procederá a establecer el enlace con la misma y cuando finalmente nos hayamos conectado, aparecerán por pantalla los controles de la figura 3.6.

Fig. 3.6 Interfaz - Conectado

En ella podemos ver los siguientes controles: joystick, switch, sliders y botones cuyo funcionamiento se detallará más adelante.

Podemos observar el botón desconexión, que servirá como su nombre indica, para desconectarnos de la plataforma.

Además entre los tres controles de la izquierda y el joystick, podemos ver un espacio en blanco. En este espacio aparecerá una flecha que le indique al usuario cual es el movimiento actual de la plataforma.

3.2.4 Joystick

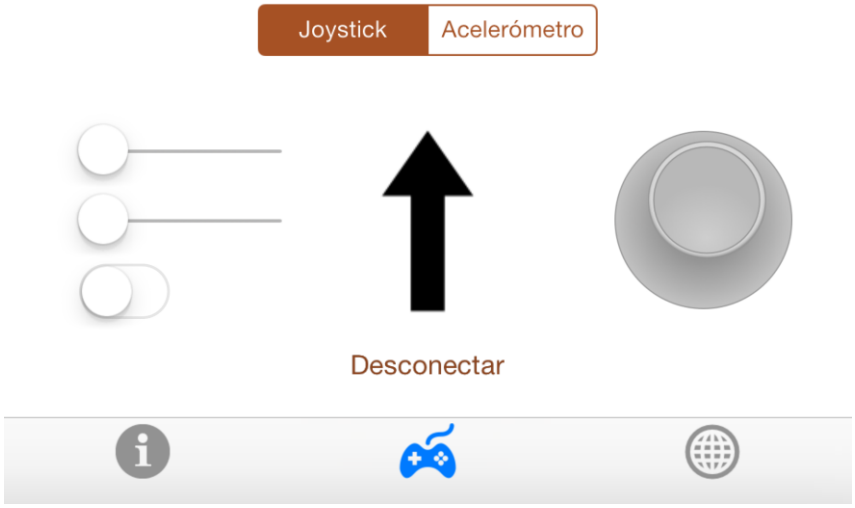


Fig. 3.7 Interfaz - Joystick

El joystick se controla pulsando el círculo de la derecha y deslizando el dedo sobre su superficie. Se puede apreciar en la imagen como el cursor se ha actualizado y esta mostrándole al usuario que se está desplazando hacia delante.

3.2.5 Acelerómetro

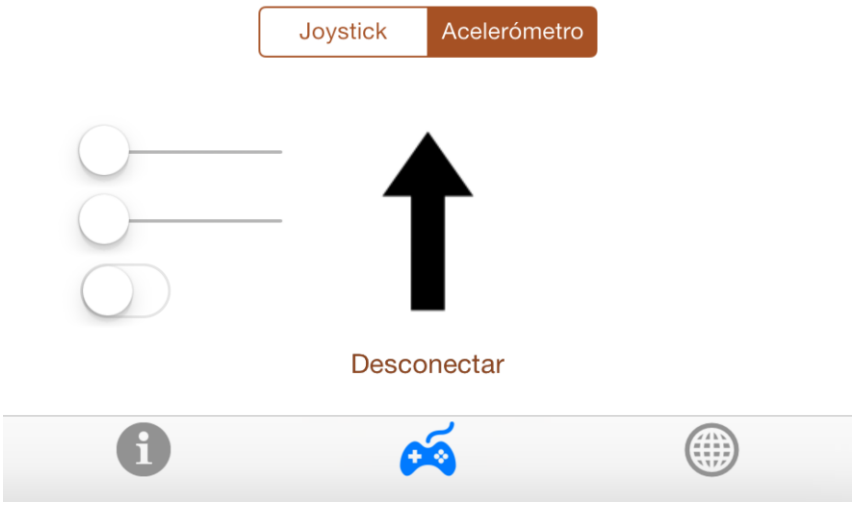


Fig. 3.8 Interfaz - Acelerómetro

Seleccionando la opción “Acelerómetro” en el menú superior, se oculta el joystick y se habilita el control de la plataforma mediante el sensor de inclinación del iPhone.

Está calibrado de manera que el punto muerto de la plataforma no se encuentre al colocar el iPhone totalmente plano, sino ligeramente inclinado hacia el usuario, debido a que esta es la posición más natural para el usuario para sujetar el dispositivo.

3.2.6 Actuadores – Sliders y switches

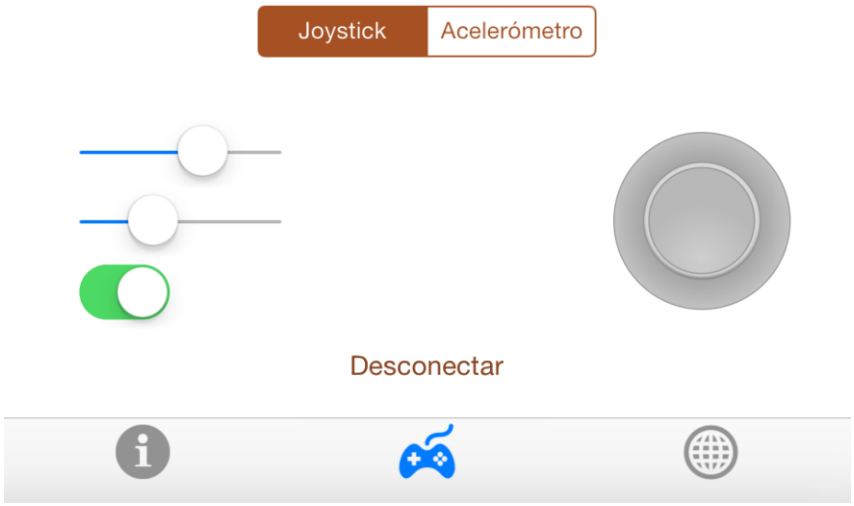


Fig. 3.9 Interfaz - Actuadores

A la izquierda de la pantalla podemos apreciar tres controles. Se corresponden con dos sliders (barras de control), destinados a controlar las señales PWM asociadas a dos servos instalados en la plataforma, y con un switch (interruptor), destinado a controlar el encendido y apagado de un led.

3.2.7 Web

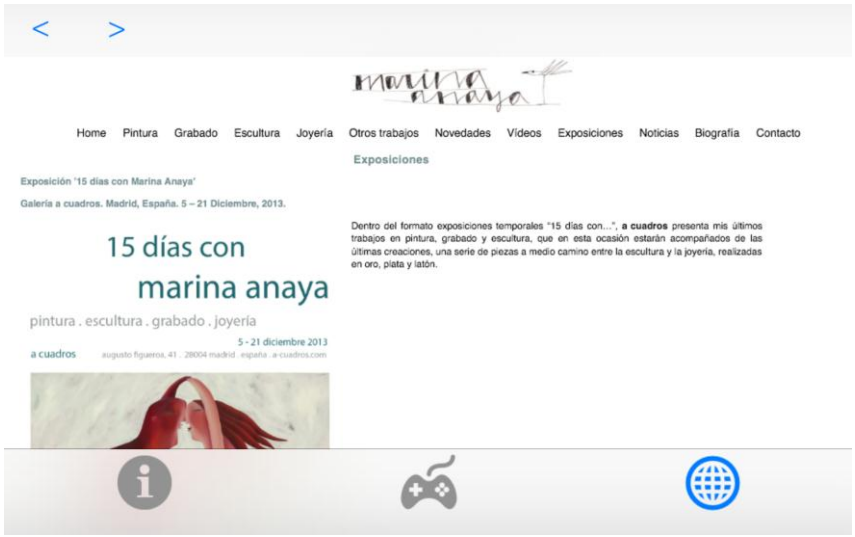


Fig. 3.10 Interfaz - Web

Se ha procedido a realizar un simple explorador web que re direccione al usuario directamente a la web de la artista. El explorador sólo dispondrá de dos botones para navegar entre las páginas visitadas.

3.2.8 Batería baja

Fig. 3.11 Interfaz - Batería baja

Debido a que las plataformas van a estar expuestas durante un largo período de tiempo en un lugar público, no se puede estar controlando constantemente la carga de la batería, por lo que se ha procedido de distintas maneras para avisar al responsable de la instalación y controlar que las baterías no se descarguen por

completo. Uno de los métodos que se ha adoptado es mostrar por pantalla al usuario que desee conectarse al dispositivo el mensaje de alerta que se puede ver en la figura 3.11[5].

3.2.9 Diagrama de flujo

A continuación se muestra el diagrama de flujo principal de la aplicación y el diagrama de flujo de la rutina Nuevo Dato.

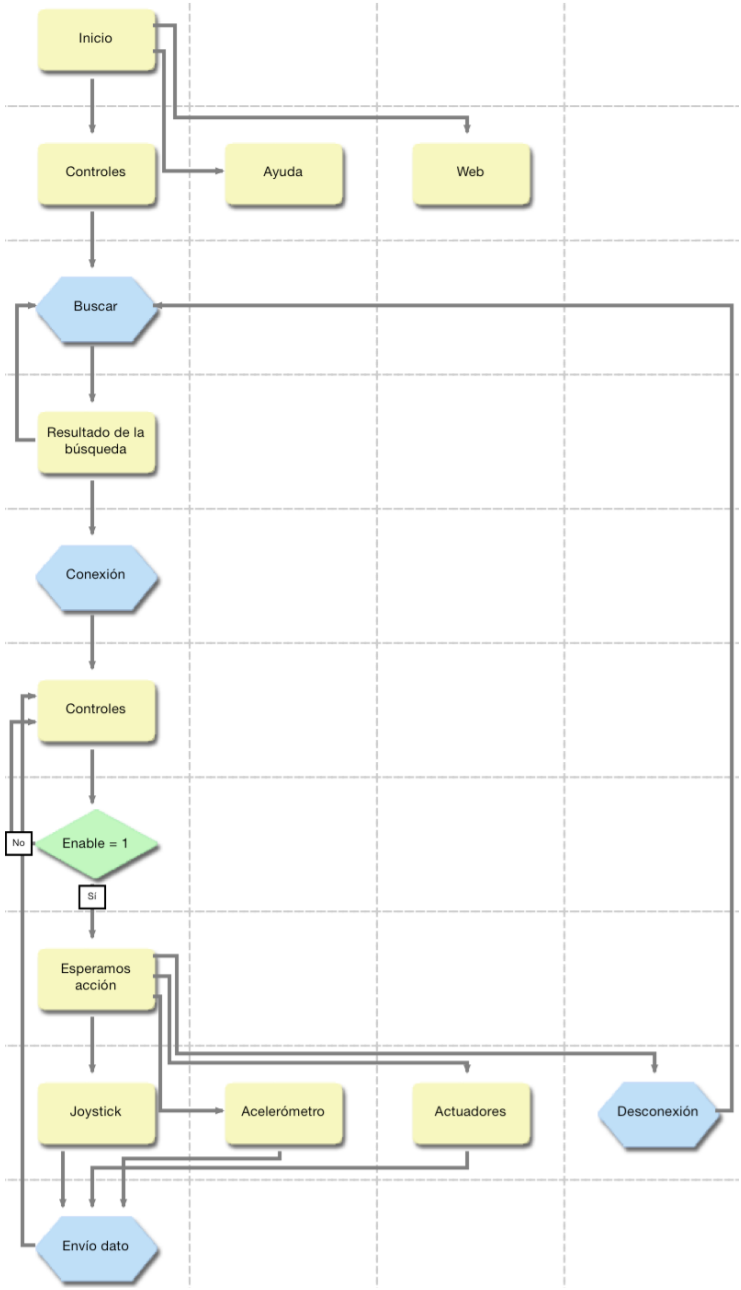


Fig. 3.12 Diagrama de flujo Xcode - Main

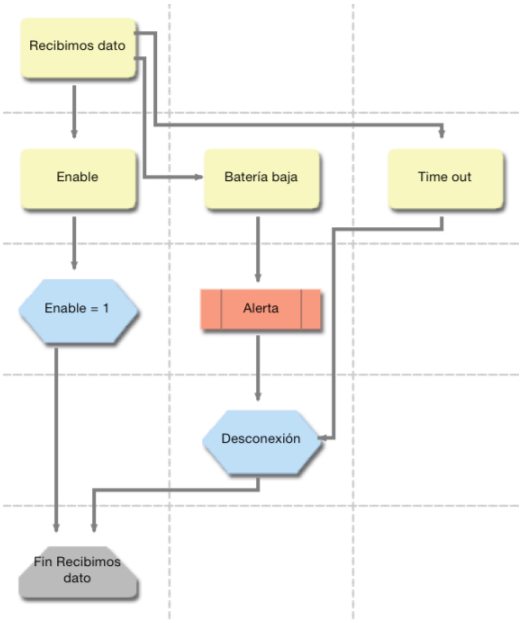


Fig. 3.13 Diagrama de flujo Xcode - Nuevo dato

3.3 Controles

A continuación se va a explicar el código asociado a cada uno de los objetos con los que interacciona el usuario durante su interacción con la aplicación.

3.3.1 Botones

La clase utilizada en Xcode que implementa el botón se llama UIButton. Este objeto intercepta las interacciones con la pantalla capacitiva y envía un mensaje a un método, mediante una conexión target-action, cuando es pulsado. Los métodos para crear esta conexión los hereda de la clase UIControl, que además le provee de métodos que permiten al usuario modificar su apariencia como el título, color, imagen, o el estado del botón[1].

A continuación se va a proceder a explicar el procedimiento para crear un objeto derivado de la clase UIControl. Éste es común para los controles: botón, slider e interruptor.

Lo primero que debemos hacer es ir al Interface Builder (Main.storyboard) y arrastrar a la representación en pantalla del iPhone un objeto button desde el menú de la derecha.

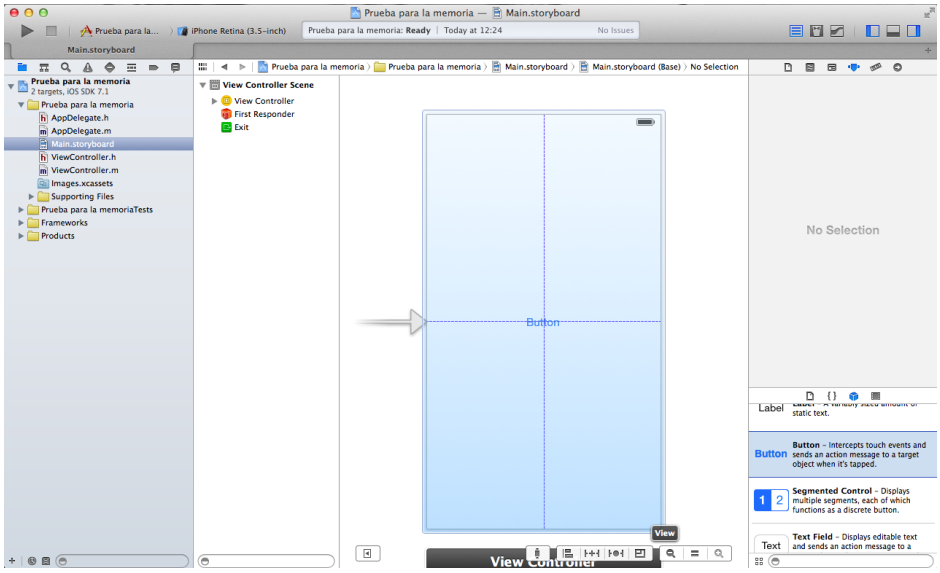


Fig. 3.14 Interface Builder - Nuevo UIButton

A continuación haremos clic en el botón "Assistant editor"  que nos permitirá ver dos archivos en pantalla.

Lo siguiente que haremos será crear la conexión target-action pulsando la tecla "ctrl" y arrastrando el botón que acabamos de crear desde Main.storyboard hasta el la implementación (ViewController.m). Se creará automáticamente un método que se llamará cada vez que el usuario pulse el botón, ejecutándose el código que hayamos escrito entre los corchetes.

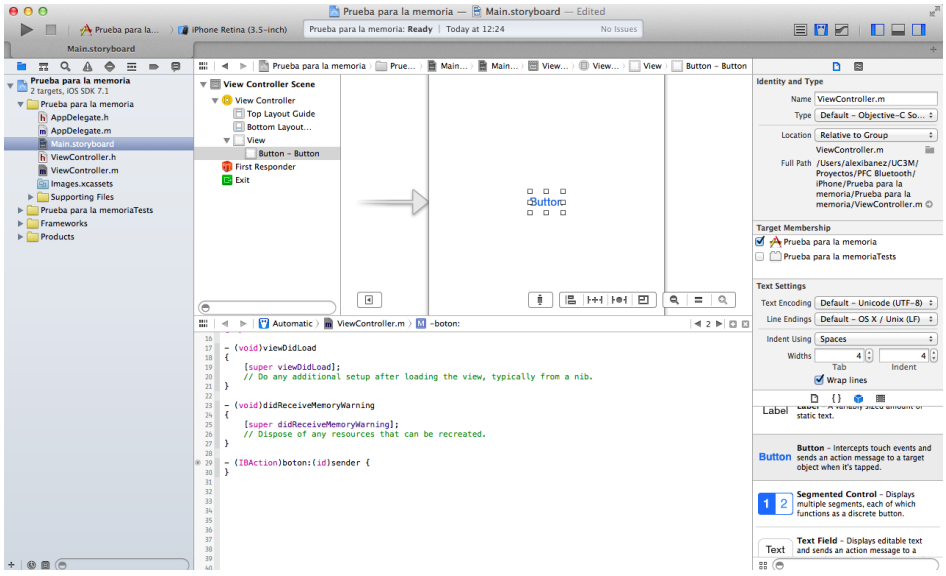


Fig. 3.15 Interface Builder - Conexión UIButton con implementación

Lo siguiente que haremos será definir el objeto en el archivo interfaz (ViewController.h) asignándole espacio en la memoria.

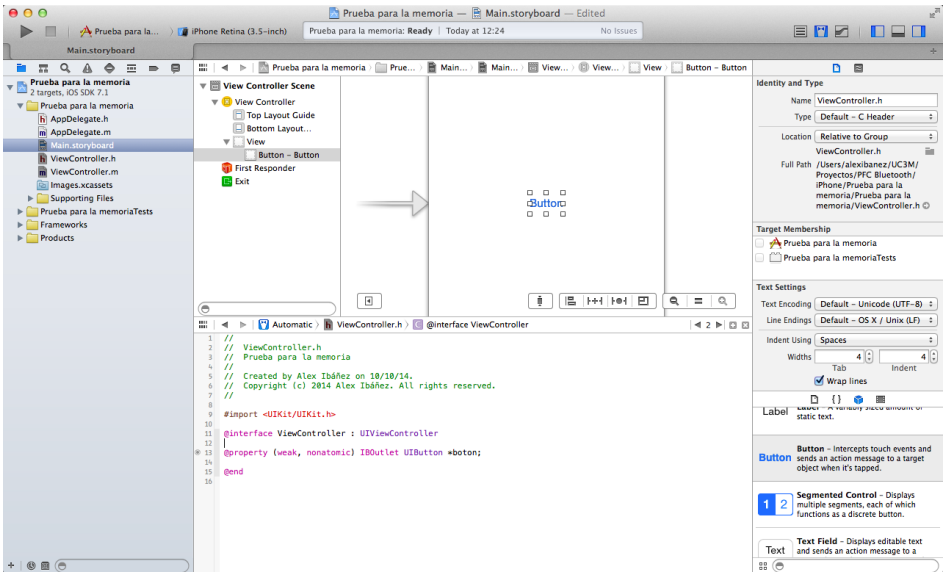


Fig. 3.16 Interface Builder - Conexión UIButton con interfaz

Una vez realizados estos tres pasos, ya tendremos declarado nuestro objeto del tipo UIControl y para programarlo y permitir al usuario interactuar con el.

Los botones que hemos declarado en este proyecto en concreto son:

```
@property (weak, nonatomic) IBOutlet UIButton *botonConectar;
@property (weak, nonatomic) IBOutlet UIButton *botonDesconectar;
@property (weak, nonatomic) IBOutlet UIButton *botonRob1;
@property (weak, nonatomic) IBOutlet UIButton *botonRob2;
```

Su código se detallará mas adelante en el apartado 5 del proyecto.

3.3.2 Sliders

Se han creado dos objetos de la clase UISlider para crear los sliders, o barras de control, que actualizarán el valor de dos señales PWM que ajustarán el ángulo de dos servos instalados en la estructura. Como hemos comentado en el apartado anterior, se declaran igual que un UIButton al ser una clase también heredada de la clase UIControl. Los objetos declarados son[1]:

```
@property (weak, nonatomic) IBOutlet UISlider *servo1;
@property (weak, nonatomic) IBOutlet UISlider *servo2;
```

3.3.3 Segmented Controller

Para permitir al usuario elegir entre el control mediante el joystick o mediante el acelerómetro se ha añadido un objeto del tipo UISegmentedControl, que

funciona como un interruptor, pero me permite declarar en el título el nombre asociado a cada estado del mismo, siendo su uso más intuitivo para el usuario[1].

```
@property (weak, nonatomic) IBOutlet UISegmentedControl *segmentedController;
```

3.3.4 Interruptores

Un interruptor de la clase UISwitch, controla el estado de un led instalado en la estructura. Su declaración es[1]:

```
@property (weak, nonatomic) IBOutlet UISwitch *led;
```

3.3.5 Joystick

El joystick no es un objeto nativo de Xcode, sino que ha sido descargado de GitHub y fue el primer programa con el que empecé a familiarizarme con el entorno de programación Xcode.

Se descargo el programa JSController del repositorio del usuario jasarien. Este programa incluía código para implementar múltiples controles, entre ellos, el joystick que fue utilizado en nuestra aplicación.

El código de este objeto se incluye en el anexo, pero básicamente son dos imágenes superpuestas. Al tocar el usuario sobre el marco de la imagen de fondo, el programa detecta el inicio de la pulsación y actualiza la posición de la imagen superior, dando la sensación de que el usuario arrastra el joystick.

3.3.6 Acelerómetro

El valor del acelerómetro lo obtenemos creando un objeto de la clase CMMotionManager.

```
@property(readonly, nonatomic) CMMotionManager *acelerometro;
```

Mediante el siguiente fragmento de código extraído del archivo implementación, inicializo el objeto, le asigno un periodo de refresco, y le asigno el método cambioValorAcelerometro que llamará cada vez que se actualice el valor de la inclinación obtenido del acelerómetro.

```
acelerometro = [[CMMotionManager alloc] init];
self.acelerometro.accelerometerUpdateInterval = .2;

[self.acelerometro startAccelerometerUpdatesToQueue:[NSOperationQueue currentQueue]
withHandler:^(CMAccelerometerData  *accelerometerData, NSError *error)
    {
        [self cambioValorAcelerometro:accelerometerData.acceleration];
        if(error) NSLog(@"%@", error);

    }];
```

4 PROGRAMACIÓN DE LA INTERFAZ

En este apartado se explica el código según se va ejecutando en función de las acciones del usuario.

4.1 Búsqueda de periféricos cercanos

Como ya se vio en el apartado anterior, lo primero que el usuario ve por pantalla al situarse en la pestaña de los controles es el botón de Búsqueda de periféricos. Al pulsarlo, se le mostrará un indicador de actividad y una barra de progreso mientras el iPhone realiza dicha búsqueda.

El código se ejecuta gracias a una conexión target-action que ya hemos explicado anteriormente.

```
- (IBAction)conectar:(id)sender {

        [bleShield.peripherals removeAllObjects];
        [bleShield findBLEPeripherals:1];

        [NSTimer scheduledTimerWithTimeInterval:(float)2.0 target:self
           selector:@selector(connectionTimer:) userInfo:nil repeats:NO];
        temporizador = [NSTimer scheduledTimerWithTimeInterval:(float)20.0
           target:self selector:@selector(timeoutTimer:) userInfo:nil
           repeats:NO];

        for (int i=0; i<5; i++) dispositivos[i]=0;

        self.botonConectar.hidden = true;
        self.botonRob1.hidden = true;
        self.botonRob2.hidden = true;
        self.spinner.hidden = false;
        self.barraProgreso.hidden = false;
        self.barraProgreso.progress = 0.25;
}
```

- Primero se eliminan los periféricos que hayan podido ser encontrados en búsquedas anteriores. Luego se llama al método `findBLEPeripherals` del objeto `bleShield` asignándole por parámetro que deje de buscar pasado 1 segundo[6].

- En las dos siguientes líneas de código se declaran dos temporizadores: el primero para iniciar una rutina en la que comprobar el número de periféricos que se han encontrado, y el segundo es un timeout de 20 segundos en caso de que ocurra algún error al intentar establecer la comunicación[6].

- En la línea siguiente inicializamos el vector llamado `dispositivos` en el que almacenaremos el número de plataformas encontradas así como el estado de utilización de cada una.

- Las seis siguientes líneas de código actualizan la información que se le muestra al usuario por pantalla.

Como hemos explicado anteriormente, a los 2 segundos se ejecuta la rutina connectionTimer.

```
-(void) connectionTimer:(NSTimer *) timer
{
    if (bleShield.peripherals.count >0)
            dispositivos[0]=bleShield.peripherals.count;
    else {
        self.botonConectar.hidden = false;
        if (temporizador) {
            NSLog(@"Invalidamos el temporizador del timeout");
            [temporizador invalidate];
            temporizador = nil;
        }
    }

    [self conexion];
}
```

- En esta rutina de atención a al interrupción del timer, comprobamos si se ha encontrado algún dispositivo y almacenamos la cuenta en el vector de dispositivos. En caso contrario invalidamos el temporizador del timeout[6].

Al final, en ambos casos se llama al método conexión.

```
-(void) conexion
{
    if (dispositivos[0] > 0){
            dispositivos[0]--;
        [bleShield connectPeripheral:[bleShield.peripherals
                objectAtIndex:dispositivos[0]]];
        self.barraProgreso.progress += 0.25;
    }
    else {
        self.barraProgreso.hidden = true;
        self.spinner.hidden = true;
        if (dispositivos[3] == 2) {
            self.botonRob1.hidden = false;
            self.botonRob1.enabled = true;
        }
        else if (dispositivos[3] == 3) {
            self.botonRob1.hidden = false;
            self.botonRob1.enabled = false;
        }
        if (dispositivos[4] == 2) {
            self.botonRob2.hidden = false;
            self.botonRob2.enabled = true;
        }
        else if (dispositivos[4] == 3) {
            self.botonRob2.hidden = false;
            self.botonRob2.enabled = false;
        }
        self.botonConectar.hidden = false;
        if (dispositivos[3] == 5 || dispositivos[4] == 5) {
```

```
            enable = true;
            self.botonConectar.hidden = true;
            self.botonRob1.hidden = true;
            self.botonRob2.hidden = true;
            self.botonDesconectar.hidden = false;
            _analogueStick.hidden = false;
            self.servo1.hidden = false;
            self.servo2.hidden = false;
            self.led.hidden = false;
            self.segmentedController.hidden = false;
            self.imagenJoystick.hidden = false;
            self.servo1.value = 0;
            self.servo2.value = 0;
            self.led.on = false;
            [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(applicationDidEnterBackground:)
name:UIApplicationDidEnterBackgroundNotification object:nil];
        }
    }
}
```

- Primero comprobamos que la cuenta de dispositivos es mayor que cero. En caso de serlo, nos conectamos al siguiente periférico de la lista y salimos de la rutina conexión.

- Si la cuenta es igual a cero, ya no quedan dispositivos por comprobar por lo que saldremos de la rutina mostrando al usuario el resultado de la búsqueda.

Una vez llamado el método connectPeripheral ahora hemos de esperar a que se establezca la conexión, invocándose el método bleConectado.

```
-(void) bleConectado
{
    if (dispositivos[3] == 4 || dispositivos[4] == 4) {
        NSLog(@"Me conecto al dispositivo");
        UInt8 buf[] = {0x06, 0x02, 0x00};
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        return;
    }
    else {
        NSLog(@"envio dato de comprobacion");
        UInt8 buf[] = {0x06, 0x01, 0x00};
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        return;
    }
}
```

- Primero comprobamos que no es el usuario el que se esta conectando al dispositivo sino que estamos consultando su disponibilidad.

- Acto seguido enviamos a la plataforma el dato de control.

Lo siguiente que esperaremos que suceda es recibir un dato de la plataforma.

```
-(void) bleNuevoDato:(unsigned char *)data length:(int)length
```

```
{
    if (temporizador) {
        NSLog(@"Invalidamos el temporizador del timeout");
        [temporizador invalidate];
        temporizador = nil;
    }
    if (data[0]==0x06 && (dispositivos[3]!=5 || dispositivos[4]!=5)) {
        if (data[1] == 0x01) {
            if (dispositivos[0] == 1) {
                dispositivos[1] = 1;
                dispositivos[2] = 0;
            }
            if (data[2] == 0x01) dispositivos[3] = 2;
            else if (data[2] == 0x02) dispositivos[3] = 3;
            else if (data[2] == 0x03) {
                dispositivos[3] = 5;
                [self conexion];
                return;
            }
        }
        else if (data[1] == 0x02) {
            if (dispositivos[0] == 1){
                dispositivos[1] = 0;
                dispositivos[2] = 1;
            }

            if (data[2] == 0x01) dispositivos[4] = 2;
            else if (data[2] == 0x02) dispositivos[4] = 3;
            else if (data[2] == 0x03) {
                dispositivos[4] = 5;
                [self conexion];
                return;
            }
        }
    }
    else if (data[0] == 0x07) {
        [self desconexion];
        return;
    }
    else if (data[0] == 0x08){
        bateria = data[1]*100+data[2];
        if (bateria <= 708) [self bateriaBaja];
        if (bateria > 708 && (dispositivos[3]==5||dispositivos[4]==5))
                    enable = true;
        return;
    }
    [[bleShield CM] cancelPeripheralConnection:[bleShield
            activePeripheral]];
    [self conexion];
}
```

- Invalidamos el temporizador del timeout[6].

- Comprobamos que el dato recibido es otro dato de control (0x06), y procesamos la información obtenida. Primero almacenamos la posición dentro del vector bleShield.peripherals en el que se encuentra el periférico actual, y después compruebo su estado de disponibilidad.

- Me desconecto del periférico y vuelvo a invocar el método conexión hasta que no haya mas periféricos.

La última vez que ejecutemos la rutina conexión, al no quedar ningún dispositivo por comprobar, se mostrará al usuario el resultado y se habrá concluido el algoritmo de búsqueda de dispositivos.

4.2 Conexión con un periférico

Una vez que el usuario ha realizado la búsqueda de dispositivos cercanos, podrá conectarse con uno de ellos pulsando el botón correspondiente, declarados en la interfaz como botonRob1 y botonRob2.

Cada botón tiene configurada una conexión target-action en la que se conecta al dispositivo en el índice del vector de periféricos determinado por dispositivos[1] y dispositivos[2] respectivamente.

```
- (IBAction)conectarRob1:(id)sender {
    [bleShield connectPeripheral:[bleShield.peripherals
        objectAtIndex:dispositivos[1]]];
    self.botonRob1.hidden = true;
    self.botonRob2.hidden = true;
    self.botonConectar.hidden = true;
    dispositivos[3]=4;
}
- (IBAction)conectarRob2:(id)sender {
    [bleShield connectPeripheral:[bleShield.peripherals
        objectAtIndex:dispositivos[2]]];
    self.botonRob1.hidden = true;
    self.botonRob2.hidden = true;
    self.botonConectar.hidden = true;
    dispositivos[4]=4;
}
```

- Nos conectamos con el dispositivo y almacenamos en dispositivos[3] o dispositivos[4] que es el usuario el que se está conectando.

Una vez que el iPhone ha establecido la comunicación, se ejecuta el método bleConectado, pero esta vez enviaré el dato de conexión en vez de el de control.

```
-(void) bleConectado
{
    if (dispositivos[3] == 4 || dispositivos[4] == 4) {
        NSLog(@"Me conecto al dispositivo");
        UInt8 buf[] = {0x06, 0x02, 0x00};
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        return;
    }
    else {
        NSLog(@"envio dato de comprobacion");
        UInt8 buf[] = {0x06, 0x01, 0x00};
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        return;
    }
```

```
}
```

Ahora esperaré la respuesta de la plataforma que me confirme que la conexión se ha realizado al recibir un dato nuevo que analizaré mediante la rutina bleNuevoDato.

La primera acción que se ejecutará sobre la plataforma será medir el nivel de batería, de manera que si ésta está descargada, se avise al usuario para que a su vez éste se ponga en contacto con el responsable de la instalación y se cambie/recargue la batería[5].

Una vez realizada la comprobación de la batería, se mostrarán todos los controles de la plataforma de manera que el usuario ya pueda interactuar con la plataforma.

4.3 Comunicación con un periférico

Ahora la interfaz detectará las acciones que el usuario ejecute sobre la misma, y enviará a la plataforma móvil el mensaje correspondiente a cada acción. Como se ha explicado antes, hay un método asociado a cada conexión target-action, y en cada uno de éstos se procesará individualmente cada acción[2].

4.3.1 Enviar dato – Joystick

El joystick no es un objeto nativo de Xcode, sino que se ha programado independientemente. Cada vez que se actualiza su valor, se llama el método:

```
- (void)analogueStickDidChangeValue:(JSAnalogueStick *)analogueStick
{
        [self actualizarJoystick];
}
```

Cada vez que se llame este método nosotros ejecutaremos actualizarJoystick.

```
- (void) actualizarJoystick
{
  if (enable){

    UInt8 buf[] = {0x01, 0x00, 0x00};
    buf[1]=(self.analogueStick.yValue*50 + 50) +
            (self.analogueStick.xValue*25);
    buf[2]=(self.analogueStick.yValue*50 + 50) –
            (self.analogueStick.xValue*25);

       if (buf[1] < 0 || buf[1] > 200) buf[1]=0;
       if (buf[2] < 0 || buf[2] > 200) buf[2]=0;
       if (buf[1] > 100) buf[1]=100;
       if (buf[2] > 100) buf[2]=100;

    NSData *data = [[NSData alloc] initWithBytes:buf length:3];
    [bleShield write:data];
    parada = false;
    enable = false;
    }
```

```
    NSString *flecha;
    if (self.analogueStick.yValue<0.2&self.analogueStick.yValue>-0.2) {
        if (self.analogueStick.xValue > 0.2) flecha = @"rotad";
        else if (self.analogueStick.xValue < -0.2) flecha = @"rotai";
        else flecha = @"stop";
    }
    if (self.analogueStick.xValue<0.2&self.analogueStick.xValue>-0.2) {
        if (self.analogueStick.yValue < -0.2) flecha = @"abajo";
        else if (self.analogueStick.yValue > 0.2) flecha = @"arriba";
        else flecha = @"stop";
    }
    if (self.analogueStick.yValue > 0.2) {
        if (self.analogueStick.xValue < -0.2) flecha = @"arribai";
        else if (self.analogueStick.xValue > 0.2) flecha = @"arribad";
        else flecha = @"arriba";
    }
    if (self.analogueStick.yValue < -0.2) {
        if (self.analogueStick.xValue < -0.2) flecha = @"abajoi";
        else if (self.analogueStick.xValue > 0.2) flecha = @"abajod";
        else flecha = @"abajo";
    }

    UIImage *img = [UIImage imageWithContentsOfFile:[[NSBundle
            mainBundle] pathForResource:flecha ofType:@"png"]];
    [_imagenJoystick setImage:img];
}
```

- Primero cambiaremos los datos, pasándolos de un rango de entre -1.0 y 1.0 a un rango de 0 a 100.

- Luego enviaremos los datos de cada eje del joystick en un byte sin signo.

- Por último actualizaremos la imagen central en función de la dirección y sentido de movimiento que esté realizando la plataforma.

4.3.2 Enviar dato – Acelerómetro

La rutina que controla el valor del acelerómetro se actualiza con una alta frecuencia, y la vamos a emplear para resetear el valor de los motores cuando levantemos el dedo del joystick, ya que, debido a su programación, el último valor enviado se corresponde con la última posición del joystick en vez de enviar la posición correspondiente al origen de coordenadas del joystick (0,0).

```
-(void)cambioValorAcelerometro:(CMAcceleration)acceleration
{
    if (_analogueStick.tocando == false && enable == true && acelOn == false && desconexion ==
false && parada == false) {
        parada = true;
        enable = false;
        UInt8 buf[] = {0x05, 0x00, 0x00};
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
    }
    if (acelOn) {
        if (enable && !desconexion){
```

```
            NSLog(@"enviando dato");
            UInt8 buf[] = {0x01, 0x00, 0x00};
            buf[1]=(-(acceleration.x-0.25)*60 + 60) +
                        (acceleration.y*25);
            buf[2]=(-(acceleration.x-0.25)*60 + 60) –
                        (acceleration.y*25);

            if (buf[1] < 0 || buf[1] > 200) buf[1]=0;
            if (buf[2] < 0 || buf[2] > 200) buf[2]=0;
            if (buf[1] > 100) buf[1]=100;
            if (buf[2] > 100) buf[2]=100;

            NSData *data = [[NSData alloc] initWithBytes:buf length:3];
            [bleShield write:data];
            parada = false;
            enable = false;
        }
        NSString *flecha;
        if (acceleration.x < maxX & acceleration.x > minX) {
            if (acceleration.y > maxY) flecha = @"rotad";
            else if (acceleration.y < minY) flecha = @"rotai";
            else flecha = @"stop";
        }
        if (acceleration.y < maxY & acceleration.y > minY) {
            if (acceleration.x > maxX) flecha = @"abajo";
            else if (acceleration.x < minX) flecha = @"arriba";
            else flecha = @"stop";
        }
        if (acceleration.x < minX) {
            if (acceleration.y < minY) flecha = @"arribai";
            else if (acceleration.y > maxY) flecha = @"arribad";
            else flecha = @"arriba";
        }
        if (acceleration.x > maxX) {
            if (acceleration.y < minY) flecha = @"abajoi";
            else if (acceleration.y > maxY) flecha = @"abajod";
            else flecha = @"abajo";
        }

        UIImage *img = [UIImage imageWithContentsOfFile:[[NSBundle
                    mainBundle] pathForResource:flecha ofType:@"png"]];
        [_imagenJoystick setImage:img];
    }
}
```

- Se comprueba si el usuario esta controlando el dispositivo mediante el acelerómetro gracias a la variable `acelOn`. En caso de ser así, se recoge el dato de la inclinación, y se cambia su rango de entre -1.0 y 1.0 a un rango entre 0 y 100.

- Se envía el dato de cada eje del acelerómetro en un byte sin signo.

- Por último actualizaremos la imagen central en función de la dirección y sentido de movimiento que esté realizando la plataforma.

4.3.3 Enviar dato – Actuadores

Cada Actuador tiene su propio método desde el que envía el valor a la plataforma.

```
- (IBAction)servo1:(id)sender {
    if (enable) {
        UInt8 buf[] = {0x02, 0x00, 0x00};
        buf[1]=(UInt8)_servo1.value;
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        enable = false;
    }
}

- (IBAction)servo2:(id)sender {
    if (enable) {
        UInt8 buf[] = {0x03, 0x00, 0x00};
        buf[1]=(UInt8)_servo2.value;
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        enable = false;
    }
}

- (IBAction)led:(id)sender {
    if (enable) {
        UInt8 buf[] = {0x04, 0x00, 0x00};
        if (_led.isOn) buf[1] = 0x01;
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        enable = false;
    }
}
```

4.3.4 Recibir dato

Ya hemos visto el método bleNuevoDato anteriormente, mientras nos conectábamos con un periférico. Ahora vamos a ver que sucede al recibir distintos mensajes desde la plataforma[2].

```
-(void) bleNuevoDato:(unsigned char *)data length:(int)length
{
        if (temporizador) {…}
        if (data[0]==0x06 && (dispositivos[3]!=5 || dispositivos[4]!=5)) {…}
        else if (data[0] == 0x07) {
      [self desconexion];
      return;
        }
        else if (data[0] == 0x08){
      bateria = data[1]*100+data[2];
      if (bateria <= 708)
                          [self bateriaBaja];
      if (bateria > 708 && (dispositivos[3]==5 || dispositivos[4]==5))
                          enable = true;
          return;
      }
}
```

- El byte 0x06, como hemos visto con anterioridad, es el byte de control.

- El byte 0x07 lo envía la plataforma cuando se supera el tiempo de espera (stand by) de la misma (60 segundos de inactividad), para realizar la desconexión con el iPhone y liberar la plataforma para que pueda ser usada por otro usuario.

- El byte 0x08 lo envía la plataforma cuando ha procesado el último dato enviado y esta lista para recibir un nuevo dato. Esto se controla internamente mediante la variable de instancia enable.

4.4 Desconexión

Hay 3 formas distintas de desconectarnos de la plataforma: por medio de un botón en la interfaz, al superar un tiempo máximo de inactividad, o al salir de la aplicación. Los dos primeros siguen el mismo algoritmo, y el tercero, debido a que la cantidad de código que se puede ejecutar al salir de la aplicación es menor, se diferente[2] [3].

4.4.1 Desconexión – Botón/timeout

Al pulsar el botón se ejecuta el siguiente método.

```
- (IBAction)desconectar:(id)sender {
    [self desconexion];
}
```

Al superar el timeout recibimos el byte 0x07 de la plataforma.

```
else if (data[0] == 0x07) {
          [self desconexion];
          return;
}
```

Ambas rutinas llaman al método desconexión. El algoritmo que sigue ahora consiste en distintos temporizadores que me permitan realizar la desconexión sin que se solape ninguna de las acciones que serán llevadas a cabo a través de la antena Bluetooth, dejando un lapso de tiempo entre una y otra para dejar que la plataforma procese cada comando[6].

```
-(void) desconexion
{
    enable = false;
    acelOn = false;
    desconexion = true;
    segmentedController.selectedSegmentIndex = 0;
    self.spinner.hidden = false;
    self.barraProgreso.progress = 0.25;
    self.barraProgreso.hidden = false;
    self.botonDesconectar.hidden = true;
    _analogueStick.hidden = true;
    self.servo1.hidden = true;
    self.servo2.hidden = true;
    self.led.hidden = true;
```

```
    self.segmentedController.hidden = true;
    self.imagenJoystick.hidden = true;

    [NSTimer scheduledTimerWithTimeInterval:(float)1.0 target:self
            selector:@selector(connectionTimer2:) userInfo:nil repeats:NO];
}
```

El método desconexion quita todos los controles que tenía el usuario en la pantalla del iPhone e inicializa el temporizador connectionTimer2[6].

```
-(void) connectionTimer2:(NSTimer *) timer
{
    self.barraProgreso.progress += 0.5;
    enable = false;
    UInt8 buf[] = {0x06, 0x03, 0x00};
    NSData *data = [[NSData alloc] initWithBytes:buf length:3];
    [bleShield write:data];
    [NSTimer scheduledTimerWithTimeInterval:(float)2.0 target:self
            selector:@selector(connectionTimer3:) userInfo:nil repeats:NO];
    [self dealloc1];
}
```

Tras 1 segundo se ejecuta el método connectionTimer2 desde el cual enviamos el byte de desconexión a la plataforma. Además inicializamos el temporizador connectionTimer3[6].

```
-(void) connectionTimer3:(NSTimer *) timer
{
    [[bleShield CM] cancelPeripheralConnection:[bleShield
            activePeripheral]];
    self.barraProgreso.hidden = true;
    self.spinner.hidden = true;
    self.botonConectar.hidden = false;
    desconexion = false;
    enable = false;
    parada = false;
    for (int i=0; i<5; i++) dispositivos[i]=0;
}
```

Tras 2 segundos, cancelamos la conexión desde el método connectionTimer3, limpiando el buffer de comunicación bluetooth e inicializando el vector dispositivos.

4.4.2 Desconexión – Cierre de la aplicación

Al final del método conexion, se crea una notificación en la que se conecta el evento UIApplicationDidEnterBackgroundNotification con el método applicationDidEnterBackground.

```
[[NSNotificationCenter defaultCenter]addObserver:self selector:
@selector(applicationDidEnterBackground:)name:UIApplicationDidEnterBackgroundNotification
object:nil];
```

En el método applicationDidEnterBackground se procederá a la desconexión de la plataforma.

```
- (void)applicationDidEnterBackground:(NSNotification *)notification
{
    [self appGoingToSleep];
```

```
    desconexion = true;
    enable = false;
}

-(void) appGoingToSleep
{
    self.botonDesconectar.hidden = true;
    _analogueStick.hidden = true;
    self.servo1.hidden = true;
    self.servo2.hidden = true;
    self.led.hidden = true;
    self.segmentedController.hidden = true;
    self.imagenJoystick.hidden = true;

    if (dispositivos[3] == 5 || dispositivos[4] == 5) {
        UInt8 buf[] = {0x06, 0x03, 0x00};
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        [NSTimer scheduledTimerWithTimeInterval:(float)1.0 target:self
                selector:@selector(connectionTimer3:) userInfo:nil repeats:NO];
    }
    [self dealloc1];
}
```

Al final del método se inicia el temporizador connectionTimer3. Que se ejecutará la próxima vez que se vuelva a acceder a la aplicación[6].

```
-(void)dealloc1
{
    [[NSNotificationCenter defaultCenter] removeObserver:self
            name:UIApplicationDidEnterBackgroundNotification object:nil];
}
```

El método dealloc1 deshabilita la notificación para evitar errores de memoria.

5 DISEÑO DE LA PLATAFORMA

5.1 Introducción

La plataforma se diseña como un robot diferencial, es decir, con dos ruedas motrices en el mismo eje de acción, y dos rodamientos que le aportan estabilidad, de manera que tiene una gran movilidad pudiendo avanzar en cualquier dirección y sentido además de poder rotar sobre si misma.

La fabricación de la plataforma ha sido posible gracias al crecimiento de la tecnología de impresión 3D, que ha permitido realizar tanto el soporte como la carcasa a partir de un modelo original realizado por ordenador de manera que se ajustase perfectamente a las necesidades del proyecto.

El diseño del circuito impreso (PCB) se ha realizado mediante ordenador y ha sido fabricada por el departamento de Sistemas y Automática de la universidad. Esto ha permitido disminuir en gran medida el tamaño de la plataforma al ser capaces de realizar todas las conexiones en el menor espacio posible y con una geometría determinada que ha permitido alojar la batería en el centro de la plataforma reduciendo la altura inicial a la mitad[13] [14].

5.2 PCB

5.2.1 Introducción

En electrónica un circuito impreso o PCB (del inglés, Printed Circuit Board) es una superficie constituida por caminos o pistas de material conductor laminadas sobre una base no conductora. El circuito impreso se utiliza para conectar eléctricamente los componentes electrónicos a través de los caminos conductores, y sostenerlos mecánicamente por medio de la base. Los caminos son generalmente de cobre mientras que la base se fabrica de resinas de fibra de vidrio reforzada. En este caso se ha utilizado una PCB de doble capa con plano de masa, aprovechando así el cobre sobrante.

La PCB se ha diseñado mediante el software informático Eagle® del desarrollador Cadsoftusa. Este sw tiene dos versiones, una de pago y otra gratuita con una funcionalidad básica. Se ha utilizado el sw gratuito par la realización de este proyecto[16].

Una vez instalado, lo abrimos, creamos un nuevo proyecto y vamos añadiendo los componentes al archivo “Esquematico.sch”. Aquí creamos todas las conexiones gráficamente, uniendo ambos extremos de la via; o programáticamente, nombrándolos con el mismo identificador. Una vez creado el archivo .sch, procederemos a colocar los componentes en la que será su posición final en el archivo “Esquematico.brd”. Este archivo será el que luego exportemos

para realizar nuestra PCB a tamaño real por lo que todo se imprimirá tal y como lo diseñemos en este archivo.

Finalizado ya el proceso de diseño, en el gestor CAD/CAM de Eagle®, procederemos a crear los archivos gerber. Estos archivos son individuales para cada capa de nuestra PCB teniendo que crear cuatro de ellos: TOP, BOTTOM, DIMENSION, y DRILL. Estos archivos almacenan una serie de coordenadas que ha de seguir la fresadora CNC para la realización de la PCB[16].

Como hemos comentado anteriormente, lo primero fue el diseño del Esquematico.sch. En este diseño se abordaron las siguientes consideraciones: la placa debe tener dos reguladores de tensión diferentes, uno para la electrónica y otro para alimentar los motores; además se estudió la forma más eficiente de controlar el sentido de giro y la potencia de los motores, siendo un puente H la solución más eficiente. A continuación se detallarán los componentes y el criterio seguido para su elección.

5.2.2 Regulador de tensión (5v) – Electrónica

Se utiliza un integrado LM7805CV. Este regulador es capaz de entregar hasta 1A a una tensión de salida prefijada de 5v por lo que no hace falta ninguna resistencia de ajuste. Se han conectado dos condensadores de tántalo (condensadores de bypass) de 10 μF y 35v en los pines de entrada y salida del integrado para filtrar los picos de tensión provocados por la electrónica que podrían dañar la batería. Además se ha añadido un diodo de protección entre dichos terminales para evitar que alguno de los condensadores se descargue violentamente a través de los circuitos internos del integrado pudiendo dañarlo.

Debido a que en algunas ocasiones se exige al componente llegar a niveles de corriente cercanos a su límite operativo, el componente se calienta debido a la protección térmica interna que actúa ante sobrecargas, por lo que se le ha incorporado un disipador de calor.

Se adjunta un datasheet del integrado en el anexo.

5.2.3 Regulador de tensión (7v) – Motores

Se barajaron distintas opciones ya que los motores son los componentes que más energía requieren de la batería. Inicialmente se consideró la utilización de un convertidor de corriente continua Buck o step-down, siendo esta la opción más indicada para circuitos electrónicos de potencia, aportando además una gran eficiencia. Pero en nuestro caso la corriente consumida por los motores es, dependiendo del ciclo de trabajo de la PWM, unos 400mA en el caso más desfavorable por cada motor. Teniendo en cuenta estas consideraciones se opta por un integrado LM317T, más sencillo de utilizar y también más compacto, lo

cual también supone una característica determinante a la hora de elegir este componente.

La tensión de salida del LM317T es variable y se ajusta mediante dos resistencias externas conectadas entre el terminal de salida y el de ajuste, y entre el terminal de ajuste y tierra (GND).

La ecuación de la que depende la tensión de salida es la siguiente:

$$V_{out} = V_{ref}\left(1 + \frac{R_2}{R_1}\right) = 1.25\left(1 + \frac{R_2}{R_1}\right)$$

La tensión V_{ref} viene fijada por el fabricante a 1.25v por lo que nosotros tendremos que elegir el valor de R_1 y R_2 de entre los valores de resistencias normalizadas para obtener la tensión de salida deseada de 7v que se empleará para alimentar los motores.

$$7 = 1.25\left(1 + \frac{R_2}{R_1}\right) \qquad R_2 = 4.6 \cdot R_1$$

Finalmente los valores elegidos son:

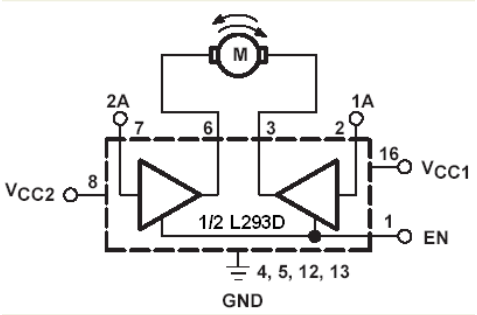
$$R_2 = 4.6M\Omega \quad R_1 = 1M\Omega$$

Se adjunta un datasheet del integrado en el anexo

5.2.4 Controlador de los motores – Puente H

Se elige un integrado L293D. Este circuito tiene múltiples configuraciones de entre las cuales elegimos la correspondiente al control de dos motores con dos sentidos cada uno. Este integrado es muy interesante porque, aunque nos permite una corriente por motor menor que el integrado L293, ya viene con los diodos de protección de contracorriente para cargas inductivas lo cual resulta en una reducción en el tamaño final del circuito impreso. Se selecciona este encapsulado debido a que el L293D nos proporciona los 400mA necesarios por motor.

El funcionamiento de un puente H consiste en 3 entradas por cada motor: un enable y dos para controlar el sentido, de manera que su funcionamiento lógico es el siguiente.



EN	1A	2A	FUNCIÓN
H	L	H	Giro a la derecha
H	H	L	Giro a la izquierda
H	L	L	Detención rápida
H	H	H	Detención rápida
L	X	X	Detención rápida

L = bajo, H = alto, X = no afecta

Fig. 5.1 Electrónica - Puente H

Como se puede comprobar, además hay otras dos entradas correspondientes a las fuentes de tensión, una para alimentar a los motores y otra para alimentar el puente H.

Se adjunta un datasheet del integrado en el anexo.

5.2.5 Diseño del circuito impreso

Una vez realizada la elección de los componentes, procedimos a estudiar el layout más eficiente para ocupar el menor espacio posible. Como ya comentamos antes, éste resulta ser con la batería en el centro de la plataforma, minimizando la altura de la misma. Esta decisión hace que el rutado de las vías sea más exigente, dando lugar a dos versiones diferentes del proyecto al haber dos etapas de desarrollo[16].

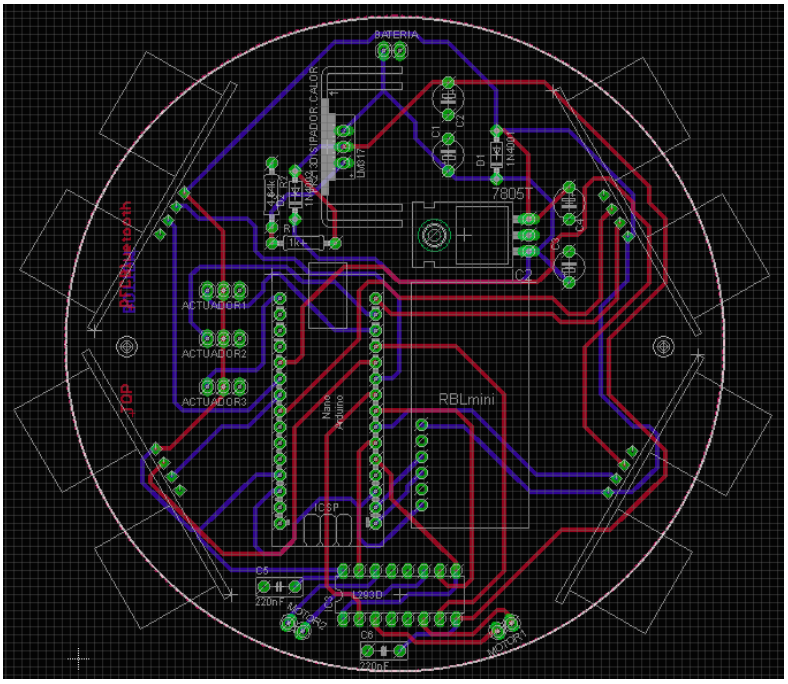


Fig. 5.2 Electrónica - PCB 1.0

La versión 1.0 se realia para empezar a probar la conexión con el iPhone y programar el Arduino con todo el Hardware listo. Fue la versión de pruebas y con la que evolucionó en proyecto.

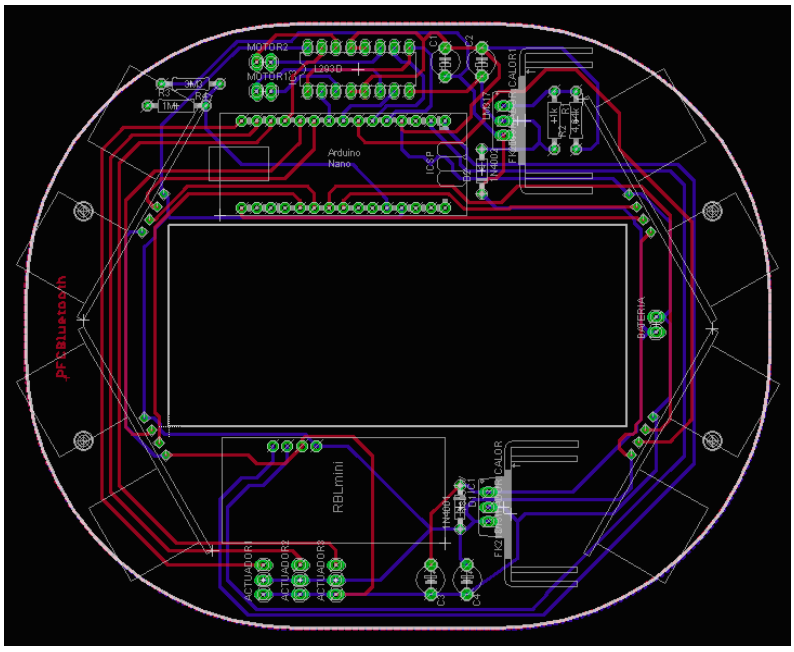


Fig. 5.3 Electrónica - PCB 2.0

La versión 2.0 se imprime, lo cual permitie optimizar el diseño y mejorarlo ya que incorpora un divisor de tensión para medir la carga de la batería, que no estaba en el primer diseño. En la fase de prueba se estudian distintas opciones para el modelo final siendo el de la figura 5.3 el definitivo.

5.3 Prototipado

5.3.1 Introducción

El prototipado es un proceso de fabricación rápido mediante la adición de material capa a capa de artículos de plástico, metal o madera. Inicialmente sólo se usaba para prototipos, y hoy en día se usa como un proceso de fabricación más. Esto se debe principalmente a la proliferación en el mercado de impresoras 3D asequibles que permiten una rápida programación y bajo coste de producción de las piezas. En este proyecto se utilizó el modelo de impresora 3D Mendel Prusa que imprime con bobinas de plástico ABS calentándolo hasta los 200º en el extrusor, y colocándolo de forma precisa, gracias a múltiples motores paso a paso, en capas consecutivas de material sobre una placa térmica calentada a 80º para evitar que la pieza se curve[13] [14].

5.3.2 Funcionamiento

Se procede con el diseño de los modelos 3D de la plataforma con la versión académica del Software Catia® de Dassault Systems. Luego se importan los archivos .stl al programa gratuito ReplicatorG, que se conecta a la impresora vía USB para cargar el conjunto de puntos que ésta debe seguir, las características de impresión, la velocidad del extrusor, la densidad de la pieza,… y otras características seleccionadas específicamente para cada pieza[14] [15].

5.3.3 Diseño de las piezas

Se diseñan tres piezas: el chasis, estructura interna que sostiene y aporta rigidez a la plataforma; la carcasa, superficie semitransparente con orificios para los ultrasonidos encargada de soportar la escultura; y la tapa que permita cambiar la batería desde la parte inferior de la plataforma.

La primera pieza es el chasis. Tiene 4 cilindros pasantes que se utilizan para soportar la tapa de la batería y la batería. Además tiene 14 taladros, 6 de los cuales se utilizan para los dos rodamientos, sobre las superficies triangulares, 4 a los lados para los motores, y 4 para unir la PCB, el chasis y la carcasa.

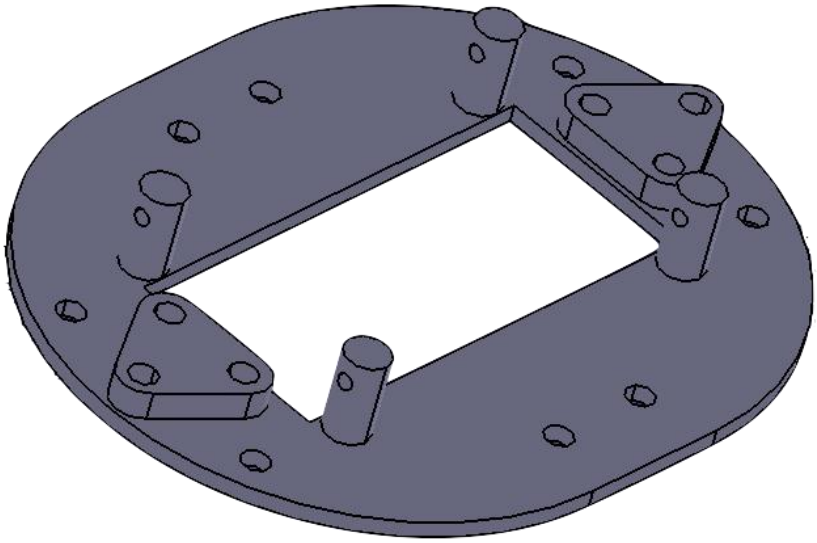


Fig. 5.4 Plataforma - Chasis

La pieza siguiente es la carcasa. Ésta tiene 8 orificios en su perímetro exterior que le permitan a los ultrasonidos emitir la onda sonora más allá de la pared de la carcasa y que esta no rebote en el interior de la plataforma. Además tiene los 4 taladros mencionados anteriormente para unirla al chasis, y 2 más en su parte más alta que la unen a la tapa sobre la que estará instalada la escultura.

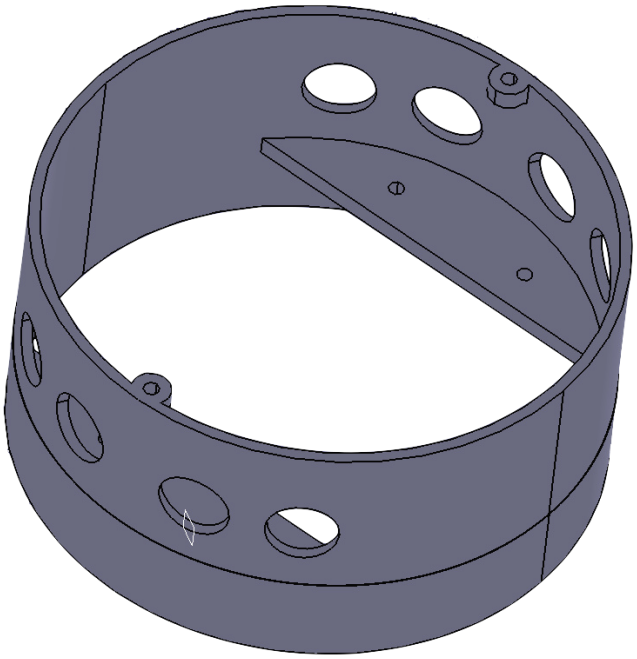


Fig. 5.5 Plataforma - Carcasa

Por último está la tapa de la batería. Esta tiene 4 orificios por los que se introducen los cilindros pasantes del chasis quedando fijada gracias a otros 4 orificios realizados en dichos cilindros a través de los cuales se insertan unas bridas que mantienen la tapa en su posición.

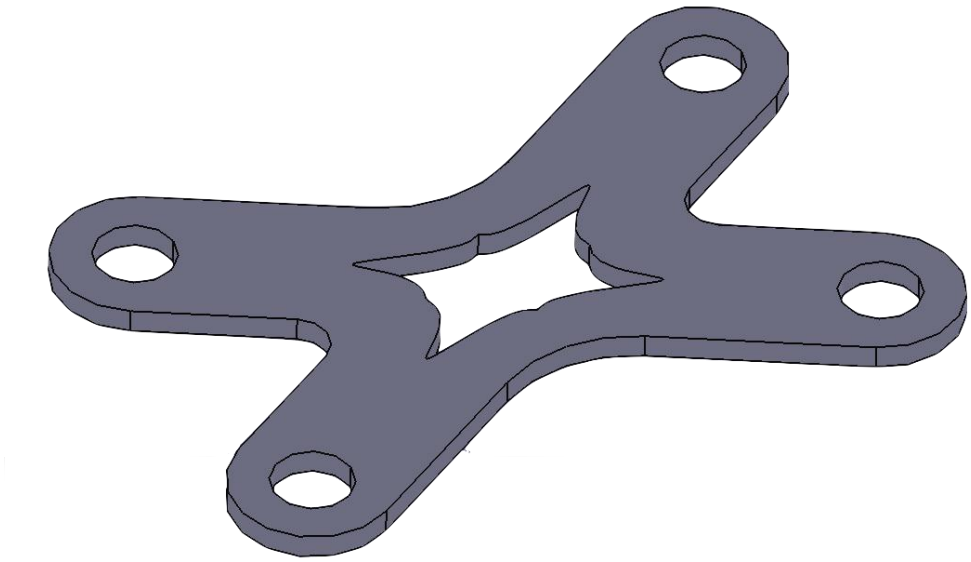


Fig. 5.6 Plataforma - Tapa de la batería

5.4 Ruedas motrices

Se selecciona un conjunto rueda-motor-reductor de la marca Pololu® muy utilizado en proyectos de electrónica.

5.4.1 Motor

Se necesita un sistema compacto y pequeño por lo que para la motorización de la plataforma se opta por un motor de corriente continua, lo cual determinará el diseño del resto de la electrónica, tanto la elección del integrado LM317, como la del puente H.

5.4.2 Reductor

Dada la falta de potencia de estos motores tan pequeños, el fabricante incorpora de fábrica el motor con el reductor en distintas desmultiplicaciones, disminuyendo así su velocidad y aumentando el par y la potencia.

El conjunto elegido es un micromotor de metal con una reductora de 30:1 con las siguientes características:

- Dimensiones: 24 x 10 x 12 mm
- Ratio de la reductora: 30:1
- Diámetro del eje: 3,9mm (con ranura de bloqueo)
- Voltaje nominal: 7Vcc (puede funcionar entre 3 a 9Vcc)
- Velocidad de giro sin carga: 440rpm
- Consumo sin carga: 40mA (Max: 360mA)
- Torque: 0,3 kg-cm (max)
- Peso: 10 gramos

5.4.3 Rueda

El mismo fabricante del conjunto motor-reductor tiene distintas ruedas que se adaptan perfectamente a la forma del eje de los mismos. Se elige una rueda pequeña de 32mm de diámetro para que no se vea demasiado afectada la velocidad de la plataforma y, otra vez, para reducir el tamaño al máximo.

En el segundo diseño de la PCB se opta por una geometría ovalada de manera que las ruedas sobresalen por encima de la base del chasis reduciendo la altura unos 5mm.

5.5 Actuadores

Vamos a analizar ahora el conjunto de componentes elegidos para que el usuario pueda interactuar con la plataforma.

5.5.1 Servo

Este dispositivo es muy utilizado en proyectos de electrónica debido a su gran versatilidad. Tiene la capacidad de ubicarse en cualquier posición dentro de su rango de acción, y de mantenerse estable en dicha posición. Esta formado por un motor de corriente continua, una caja reductora y un circuito de control que consta normalmente de un potenciómetro que realimente la posición del servo.

Hay distintos tipos de movimientos que se pueden reproducir con un servo, a la vez que hay distintos mecanismos y tamaños de servos, pero para esta aplicación se han elegido servos electrónicos analógicos de movimiento circular o angular.

Fig. 5.7 Electrónica - Servo

5.5.2 Led

Para darle un mayor atractivo al conjunto, y así aprovechar todos los pines del microcontrolador, se le añade un led a la escultura.

5.6 Sensores

No es un proyecto que predomine por el uso de sensores, de hecho hay muy poca realimentación del medio a la plataforma además de la información que el usuario le aporta mediante el iPhone. Pero para que no se hiciese un uso incorrecto de la misma, se estipula en los objetivos iniciales que llevaría distintos sensores de ultrasonidos para evitar que las esculturas choquen con las paredes y puedan dañarse.

5.6.1 Ultrasonidos

Estos sensores emiten un tren de pulsos a una frecuencia determinada imperceptible para el oído humano, y tienen un sensor capaz de filtrar las señales acústicas y quedarse con aquellas de la misma frecuencia que la emitida. Conociendo la frecuencia del reloj del microcontrolador, la velocidad a la que se propaga la onda sonora por el medio, y el número de ciclos que tarda en volver la señal, podemos calcular, con una precisión suficiente para nuestra aplicación, la distancia a un objeto cercano donde haya rebotado nuestra señal acústica.

El funcionamiento de estos sensores se ve afectado al funcionar varios en la misma habitación por lo que se debe controlar que nunca haya dos sensores emitiendo al mismo tiempo. Esto conllevará a que cuando ambas plataformas se encuentren juntas se produzca alguna interferencia, pero que se solucionará dirigiendo la plataforma en sentido opuesto a donde se encuentre la otra, o haciéndola rotar sobre sí misma ya que en este movimiento la plataforma no comprueba la distancia a objetos cercanos.

Los ultrasonidos vienen instalados en una placa y tienen 4 pines, dos para la alimentación: VCC y GND, y dos para medir la distancia: ECHO y TRIGER. El pin Triger es el que envía el tren de pulsos y el pin Echo se pondrá a nivel alto cuando lo reciba.

Fig. 5.8 Electrónica - Ultrasonidos

5.7 Ensamblaje final

El resultado final de la plataforma una vez se han soldado todos los componentes y se ha unido la PCB al chasis, es el siguiente.

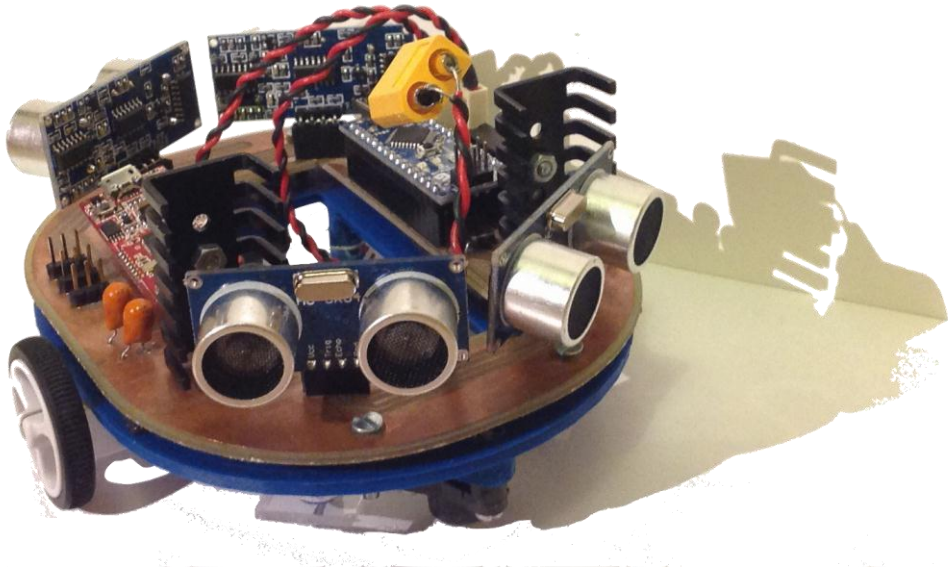


Fig. 5.9 Plataforma - Foto 1

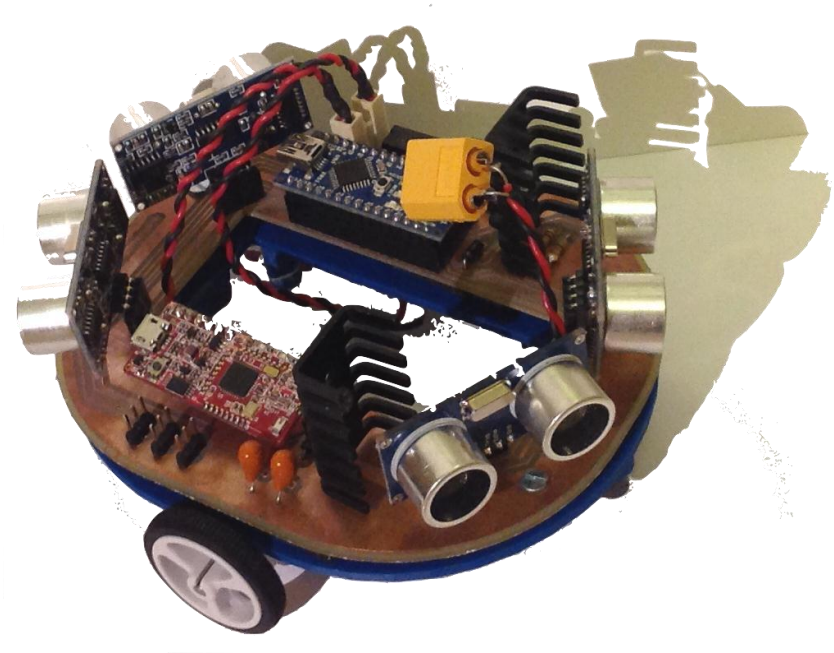


Fig. 5.10 Plataforma - Foto 2

El aspecto que tiene la plataforma con la escultura de Marina Anaya en la parte superior es el siguiente.

Fig. 5.11 Escultura de Marina Anaya

6 PROGRAMACIÓN DE LA PLATAFORMA

6.1 Arduino

6.1.1 Introducción

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares[11].

El hardware consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Los microcontroladores más usados son el Atmega168, Atmega328, Atmega1280, ATmega8 por su sencillez y bajo coste que permiten el desarrollo de múltiples diseños. Por otro lado el software consiste en un entorno de desarrollo (IDE) que implementa el lenguaje de programación Processing/Wiring y el cargador de arranque que es ejecutado en la placa[11].

Para este proyecto utilizamos la placa Arduino Nano 3.1, una de las más pequeñas dentro del conjunto de placas Arduino, que se ajusta perfectamente a las necesidades del proyecto por su reducido tamaño y por el amplio número de puertos que posee. Se basa en un microcontrolador Atmega328 cuyas características se detallan más adelante[11].

6.1.2 Especificaciones

Especificaciones

Microcontrolador	ATmega328
Tensión operativa (nivel lógico)	5 V
Tensión de entrada (recomendada)	7-12 V
Tensión de entrada (límites)	6-20 V
Pines digitales I/O	14 (de los cuales 6 son PWM)
Pines analógicos I	8
Corriente (DC) por pin I/O	40 mA
Memoria flash	32 KB (ATmega328) de los cuales 2 KB son usados por el bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad del reloj	16 MHz
Dimensiones	0.73" x 1.70"

Alimentación

El Arduino Nano puede ser alimentado a través de la conexión USB Mini-B, con una fuente de alimentación externa no regulada entre 6-20V (pin 30), o una fuente de alimentación externa de 5V regulada (pin 27). La fuente de alimentación se selecciona automáticamente a la fuente de tensión más alta.

El chip FTDI FT232RL del Nano sólo se enciende si la tarjeta está siendo alimentada a través del cable USB. Como resultado, cuando recibe alimentación externa (no USB), la salida de 3.3V (que es suministrada por el chip FTDI) no está disponible y los LEDs RX y TX parpadearán sólo si los pines digitales 0 o 1 están en nivel alto.

Memoria

El ATmega328 tiene 32KB, (de los cuales 2 KB son utilizados por el bootloader). El ATmega328 tiene 2 KB de SRAM y 1 KB de EEPROM (que pueden ser leídos y escritos con la librería EEPROM).

Entradas y Salidas

Cada uno de los 14 pines digitales en el Nano se puede utilizar como una entrada o salida, utilizando las funciones pinMode(), digitalWrite(), y digitalRead(). Operan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia de pull-up (desconectada por defecto) de 20-50 kOhms. Además, algunos pines tienen funciones especiales:

- Puerto Serie: 0 (RX) y 1 (TX). Se utiliza para recibir (RX) y transmitir (TX) datos en serie TTL. Estos pines están conectados a los pines correspondientes del USB-to-TTL chip, de la serie FTDI.

- Interrupciones externas: 2 y 3 Estos pines pueden ser configurados para activar una interrupción en un valor bajo, un flanco ascendente o descendente, o un cambio en el valor.

- PWM: 3, 5, 6, 9, 10, y 11. Proporcionan una salida PWM de 8 bits con la función analogWrite().

- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estos pines soportan la comunicación SPI, que, aunque esta soportada por el hardware, actualmente no está incluida en el lenguaje Arduino.

- LED: Hay un diodo integrado conectado al pin digital 13. Cuando el pin tiene un valor alto, el LED está encendido, cuando el valor es bajo, estará apagado.

- El Nano tiene 8 entradas analógicas, cada una de las cuales proporcionan 10 bits de resolución (es decir, 1024 valores diferentes). Por defecto se miden desde GND a 5 voltios, aunque es posible cambiar el extremo superior de su rango mediante la función theanalogReference(). Los pines analógicos 6 y 7 no se pueden utilizar como pines digitales.

- I2C: 4 (SDA) y 5 (SCL). Apoyo I2C de comunicación que utiliza la librería Wire.

- AREF. Voltaje de referencia para las entradas analógicas. Se utiliza con analogReference().

- Reset. Al poner este pin en nivel bajo, se reinicia el microcontrolador. Normalmente se utiliza para añadir un botón de reinicio a las placas de Arduino.

6.1.3 Programación

Se incluye el código comentado en el anexo.

6.1.4 Diagrama de flujo

Diagrama de flujo del programa PFCBluetooth.ino ejecutado en el Arduino. El IDE de Arduino viene programado por defecto con dos rutinas, la rutina setup en la que inicializamos la plataforma y la comunicación con la antena, y la rutina loop que equivale a un bucle infinito.

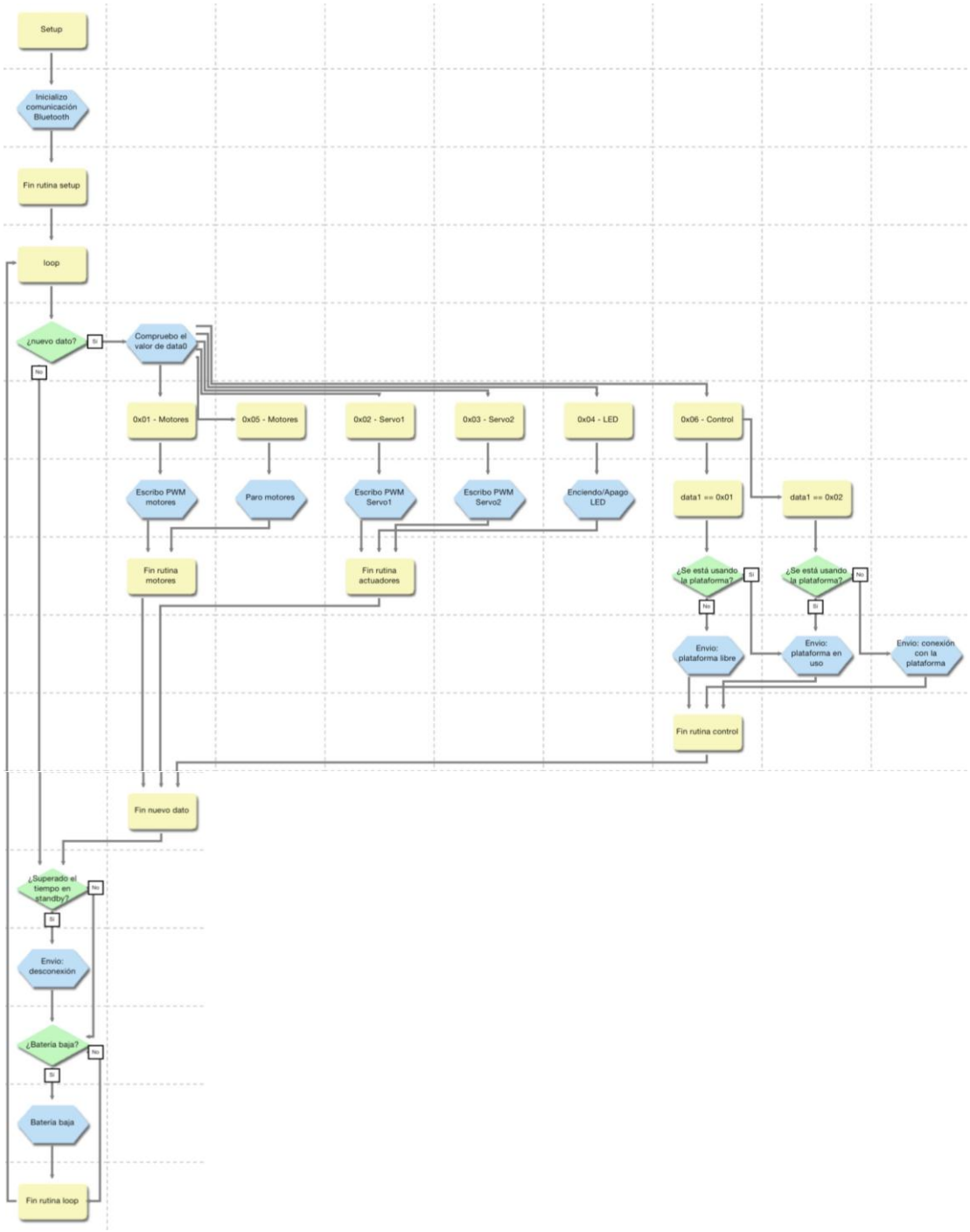


Fig. 6.1 Diagrama de flujo Arduino - Main

6.2 BLE Mini

En este apartado vamos a documentar la antena Bluetooth que se utiliza en la comunicación con imágenes y datos proporcionados por el fabricante[12].

6.2.1 Características

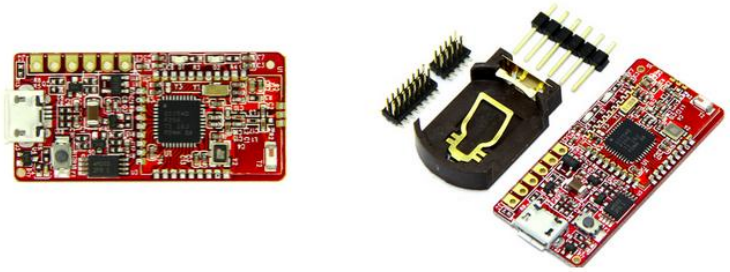


Fig. 6.2 BLE Mini - Características

Fácil de usar – Gracias al firmware pre-cargado en la antena, ésta se puede conectar fácilmente a cualquier placa con una interfaz UART.

Compacto – Con un tamaño de (L)39mm x (W)18.5mm x (H)3.8mm, el BLE Mini se ajusta al tamaño de otras placas de gran éxito como Arduino Mini/Micro/Nano, para poder llevar a cabo los diseños más compactos.

Funciona como Central o como Periférico – La antena se puede programar como central para comunicarse con otra antena de RedBearLab y no sólo con un Smartphone.

Funcionamiento autónomo - Texas Instruments (TI) CC2540 es un SoC (System-on-Chip) IC que se puede operar sin ningún tipo de microcontrolador externo. Se puede desarrollar firmware propio usando el compilador IAR Embedded Workbench para 8051. Todos los pines I/O del CC2540 son accesibles.

Firmware actualizable – Se puede desarrollar firmware propio o descargarlo del la página web de RedBearLab y subirlo a la antena BLE Mini a través del conector USB sin necesidad de utilizar un depurador TI CC.

Posibilidad de conectar una antena externa para ampliar su rango de acción.

6.2.2 Diseño

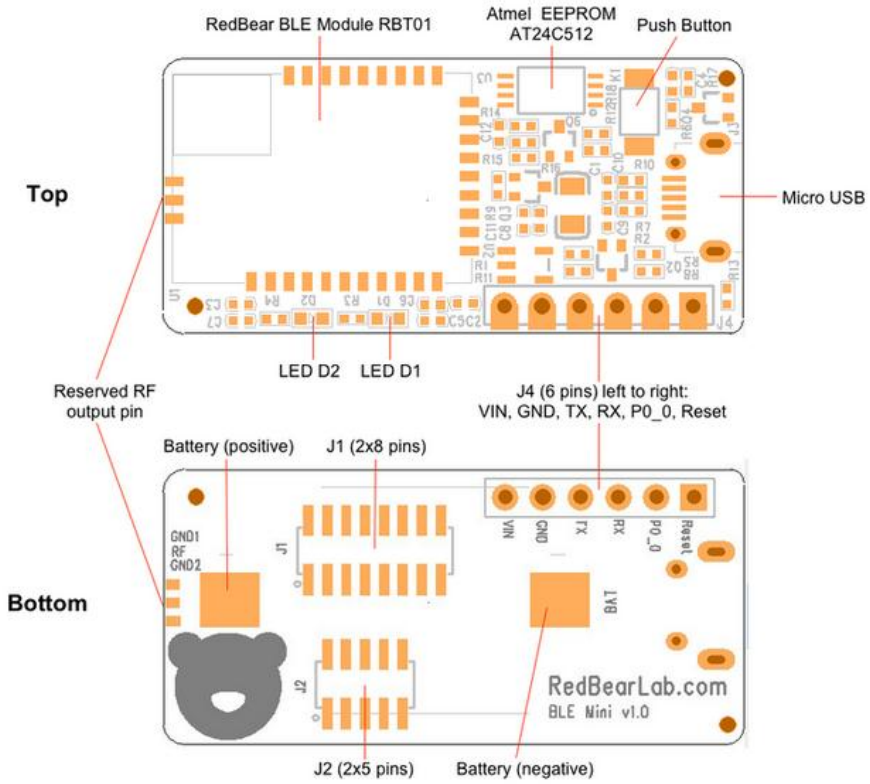


Fig. 6.3 BLE Mini - Diseño

- Módulo Bluetooth 4.0 Low Energy de RedBearLab (model: RBT01), Bluetooth SIG certified with on-board chip antenna
- Texas Instruments CC2540 2,4 GHz System-on-Chip Bluetooth (SoC)
- Dimensiones - (L) x 39mm (W) x 18.5mm (H) 3,8 mm
- Componentes programables instalados en la placa - 512Kb EEPROM, 2 LEDs (azul y verde), Push Button.
- Accesorios incluidos - Soporte para batería plana tipo moneda y 3 conectores
- El esquemático del BLE Mini esta disponible y es opensource.
- Opciones de alimentación - 3.4V a 11V VIN / 5V USB / batería plana (2025 o 2032) / 3.7V Li-ion

- No se debe conectar más de una fuente de energía al mismo tiempo, ya que ello dañaría la placa.

7 DESPACHO ECONÓMICO

En este apartado aparece un cálculo detallado de los costes de la plataforma, tanto de las componentes necesarios para su realización como de las actividades de soporte necesarias para su diseño.

7.1 Presupuesto

En la tabla siguiente aparece el coste desglosado de todos los componentes utilizados en la realización de una plataforma.

Componente	uds.	Precio unitario €	Precio total €
Motores DC	2	13,20	26,40
Ruedas (2 uds.)	1	5,95	5,95
Rodamientos (2 uds.)	1	5,10	5,10
Funda motor (2 uds.)	1	4,20	4,20
Arduino Nano	1	12,45	12,45
BLE Mini	1	26,60	26,60
Ultrasonidos	4	2,87	11,48
LM7805CV	1	0,59	0,59
LM317T	1	1,47	1,47
L293D	1	3,59	3,59
Resistencias	4	0,02	0,08
Diodos	2	0,04	0,08
Condensadores	4	1,14	4,56
Batería	1	8,65	8,65
ABS – plástico impresora	1/4	16,00	4,00
PCB	1	0,00	(a cargo del departamento)
TOTAL			115,20

Fig. 7.1 Presupuesto - Plataforma

Éste es el precio de producción individual de una plataforma. A éste habría que añadirle el coste de las actividades de soporte que han sido necesarias para la realización y el diseño de la plataforma.

Actividad	uds	Coste unitario	Coste total €
MacBook Pro	1	1129,00€	1129,00€
Licencia Apple development team	1	80€/año	80,00
Xcode	1	0,00€	0,00€
Horas de trabajo	310	30€/hora	9300,00€
TOTAL			10509,00 €

Fig. 7.2 Presupuesto - Actividades de soporte

7.2 Trabajos futuros

Dada la posibilidad de realizar una PCB que incluyese el integrado de Texas Instruments CC2540 en lugar de comprar la antena a RedBearLab, y de utilizar un microporcesador de Atmel en vez de comprar un Arduino; podríamos reducir el precio final de la plataforma. En este apartado no se tienen en cuenta los componentes extra que habría que añadir para el correcto funcionamiento de la nueva PCB ya que sólo se pretende demostrar, de forma orientativa, la diferencia de precio entre ambas soluciones.

Componente	Precio anterior €	Precio nuevo €	Diferencia €
Antena bluetooth	26,60	3,99	22,61
Microcontrolador	12,45	2,83	9,62
TOTAL	39,05	6,82	32,23

Fig. 7.3 Presupuesto - Trabajos futuros

Como se puede observar, en el hipotético caso de que se produjesen más plataformas, habría que estudiar cuantas unidades sería necesario producir para hacer rentable el tiempo en desarrollo de la nueva PCB.

8 CONCLUSIONES

8.1 Conclusiones sobre el proyecto

El proyecto une diferentes áreas de la ingeniería para poder crear la plataforma, y obliga a unificarlas todas para que el producto tenga éxito ya que no se pueden diseñar las partes por separado sin tener en cuenta la interacción entre ellas. Esto hizo que la metodología de trabajo fuese complicada al principio, pero en cuanto se realizó el primer diseño de la PCB, la programación del Arduino y la interfaz fue rápida, haciendo que todo el trabajo previo en el diseño electrónico, y la correcta elección de los componentes, hiciese que programar fuese una tarea más sencilla.

Al final del proyecto se ha conseguido cumplir todos los objetivos principales.

La interfaz cumple todos los requisitos definidos al inicio del proyecto permitiendo al usuario realizar múltiples operaciones sobre la plataforma. Se ha conseguido implementar tanto el control del movimiento mediante el joystick y el acelerómetro, como el control sobre los servos gracias a diferentes elementos de la librería UIControl de Xcode.

La plataforma ha pasado por dos versiones diferentes: una primera que se utilizó para programar la interfaz y el código del Arduino, y una segunda en la que se perfeccionó el diseño inicial reduciendo el tamaño y haciéndola más compacta. Que la plataforma fuese lo más reducida posible era uno de los requisitos iniciales y se ha cumplido sin comprometer la autonomía de la plataforma, ya que la necesidad de incorporar una batería de gran capacidad (y por lo tanto de gran tamaño) se ha cumplido sin comprometer la condición de que el diseño fuese compacto.

En conclusión, el objetivo de crear una plataforma móvil para soportar una escultura que pudiese ser controlada mediante un dispositivo iPhone, se ha cumplido.

8.2 Conclusiones personales

Las conclusiones respecto al proyecto son muy positivas. Se ha conseguido diseñar un producto que se ha podido llevar a cabo, gracias a la participación de Marina Anaya y a la tecnología actual.

En lo personal se pueden sacar conclusiones tanto en el aspecto académico como en el profesional.

Académicamente ha supuesto una gran satisfacción poder sintetizar varias años de estudio y conocimientos de distintas asignaturas en un único proyecto. Además he aprendido nuevos lenguajes de programación y a utilizar nuevos

programas que seguro serán importantes en mi carrera profesional. El hecho de que nunca antes hubiese programado para iPhone era una desventaja pero también una motivación ya que era uno de los objetivos personales principales por los que hice este proyecto. Esto hizo que al principio mis avances fuesen lentos, pero después todo ese trabajo confluyó al tener una interfaz funcional que fui perfeccionando y haciendo visualmente más atractiva.

9 TRABAJOS FUTUROS

Si bien el proyecto cumple los objetivos iniciales, una vez realizada la plataforma, hay aplicaciones potenciales y mejoras del modelo actual que se podrían llevar a cabo. Dentro de las posibles mejoras se encuentran:

- Intentar implementar una función mediante la cual ambas plataformas pudiesen comunicarse entre si, o estar las dos conectadas a un dispositivo maestro que las permita interactuar entre sí pudiendo conseguir que ambas plataformas se encuentren en una habitación, o que hiciesen una coreografía pre programada.

- Utilizar un micro superior, o una tarjeta que ampliase el número de señales PWM disponibles de manera que las plataformas tuviesen mayor movilidad o se pudiesen encender conjuntos de leds.

Esas mejoras son respecto a la funcionalidad de la plataforma, pero la aplicación también es susceptible de mejorarse.

- Se podría haber implementado un tercer tipo de control mediante 4 botones que permitiesen tener un control más sencillo pero robusto de la plataforma.

- Realizar una aplicación específica para iPad. Aunque Xcode nos permite portar la aplicación y ajustar la interfaz a la pantalla del iPad, no se aprovecha el potencial de tener una pantalla tan grande portando la App en lugar de realizar una exclusiva para iPad.

- Aunque el atractivo visual de la aplicación ha sido un detalle muy cuidado dado el aspecto artístico del proyecto, aún se podría mejorar este aspecto cambiando los botones, nativos del SDK de Xcode, por imágenes personalizadas. Además se podría añadir un fondo a la aplicación con un motivo similar al utilizado en las esculturas.

- Adaptación a iOS 8. Puede ser una posible mejora pero actualmente no sería positivo dado que la mayoría de usuarios aun tiene en sus dispositivos el Sistema Operativo anterior y no podrían utilizar esta aplicación.

10 REFERENCIAS

MANUALES

[1] López Hernández, Fernando. "El lenguaje Objective-C para programadores C++ y Java". http://exordio.qfb.umich.mx/archivos_pdf_de_trabajo_umsnh/Leer escribir PDF 2014/Objetive C 2014/Lenguaje Objective-C_castellano.pdf

[2] iOS Developer Library. "Core Bluetooth Framework Reference". https://developer.apple.com/library/ios/documentation/CoreBluetooth/Reference/CoreBluetooth_Framework/

[3] iOS Developer Library. "Core Bluetooth Programming Guide". https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/CoreBluetooth_concepts.pdf

DIRECCIONES DE INTERNET

[4] Bluetooth®. "Sitio web oficial del protocolo Bluetooth" http://www.bluetooth.com/Pages/Bluetooth-Home.aspx

[5] Stackoverflow. "iPhone - Popup alert message". http://stackoverflow.com/questions/1501589/iphone-popup-alert-message

[6] Stackoverflow. "iPhone - Stop NSTimer" http://stackoverflow.com/questions/3509012/how-to-stop-nstimer

[7] Stackoverflow. "Xcode - NSLog" http://stackoverflow.com/questions/5486270/xcode-using-nslog-for-debugging

[8] Stackoverflow. "Xcode - Bluetooth 4.0 peripheral list" http://stackoverflow.com/questions/10178293/how-to-get-list-of-available-bluetooth-devices

[9] Stackoverflow. "Xcode - Set peripheral notification" http://stackoverflow.com/questions/10612442/how-to-identify-a-key-press-from-a-bluetooth-low-energy-tag-on-the-iphone

[10] Stackoverflow. "Xcode - Autolayout" http://stackoverflow.com/questions/25766747/emulating-aspect-fit-behaviour-using-autolayout-constraints-in-xcode-6

[11] Arduino. "Web de Arduino. Información sobre las placas. Código libre" http://arduino.cc/en/pmwiki.php?n=Main/ArduinoBoardNano

[12] RedBearLab. "Web del fabricante Red Bear Lab. Producto RBL Mini" http://redbearlab.com/blemini/

[13] RepRap - Mendel Prusa “Web de las impresoras 3D RepRap” http://reprap.org/wiki/Prusa_Mendel/es

[14] ReplicatorG “Web del software de impresión 3D ReplicatorG” http://replicat.org/

[15] Catia - Dassault Systems® “Web del software de diseño 3D Catia®” http://www.3ds.com/es/productos-y-servicios/catia/

[16] Eagle - Cadsoftusa® “Web del software de diseño de PCB Eagle®” http://www.cadsoftusa.com/

11 ANEXOS

11.1 Anexo1: Datasheets

Los datasheets se pueden consultar en las siguientes páginas web.

11.1.1 Regulador de tensión 5v - LM7805CV

http://docs-europe.electrocomponents.com/webdocs/1282/0900766b81282455.pdf

11.1.2 Regulador de tensión 7v - LM317T

http://docs-europe.electrocomponents.com/webdocs/0780/0900766b807800d8.pdf

11.1.3 Puente H - L293D

http://docs-europe.electrocomponents.com/webdocs/0e77/0900766b80e779ea.pdf

11.2 Anexo 2: Código del iPhone

Un proyecto en Xcode consta de distintos archivos para controlar cada vista de la interfaz. En este caso tenemos 3 vistas diferentes y los archivos asociados al objeto joystick, lo cual componen un conjunto de 8 archivos, 4 interfaces y 4 implementaciones.

11.2.1 Joystick

- **JSAnalogueStick.h**

```
#import <UIKit/UIKit.h>

@class  JSAnalogueStick;

@protocol JSAnalogueStickDelegate <NSObject>

- (void)analogueStickDidChangeValue:(JSAnalogueStick *)analogueStick;

@end

@interface JSAnalogueStick : UIView{
    NSString* instance;
}

@property (nonatomic,readonly) BOOL tocando;      // Para resetear el movimiento cuando no se
toque el JS

@property (nonatomic, readonly) CGFloat xValue;
@property (nonatomic, readonly) CGFloat yValue;
```

```
@property (nonatomic, assign) BOOL invertedYAxis;

@property (nonatomic, assign) IBOutlet id <JSAnalogueStickDelegate> delegate;

@property (nonatomic, readonly) UIImageView *backgroundImageView;
@property (nonatomic, readonly) UIImageView *handleImageView;

@end
```

- **JSAnalogueStick.m**

```
#import "JSAnalogueStick.h"

#define RADIUS ([self bounds].size.width / 2)

@implementation JSAnalogueStick

- (id)initWithFrame:(CGRect)frame
{
        if ((self = [super initWithFrame:frame]))
        {
           [self commonInit];
        }

        return self;
}

- (id)initWithCoder:(NSCoder *)aDecoder
{
        if ((self = [super initWithCoder:aDecoder]))
        {
           [self commonInit];
        }

        return self;
}

- (void)commonInit
{
        _backgroundImageView = [[UIImageView alloc] initWithImage:[UIImage
imageNamed:@"analogue_bg"]];
        CGRect backgroundImageFrame = [_backgroundImageView frame];
        backgroundImageFrame.size = [self bounds].size;
        backgroundImageFrame.origin = CGPointZero;
        [_backgroundImageView setFrame:backgroundImageFrame];
        [self addSubview:_backgroundImageView];

        _handleImageView = [[UIImageView alloc] initWithImage:[UIImage
imageNamed:@"analogue_handle"]];
        CGRect handleImageFrame = [_handleImageView frame];
        handleImageFrame.size = CGSizeMake([_backgroundImageView bounds].size.width / 1.5,
                                           [_backgroundImageView bounds].size.height / 1.5);
        handleImageFrame.origin = CGPointMake(([self bounds].size.width -
handleImageFrame.size.width) / 2,
                                              ([self bounds].size.height -
handleImageFrame.size.height) / 2);
        [_handleImageView setFrame:handleImageFrame];
        [self addSubview:_handleImageView];

        _xValue = 0;
        _yValue = 0;
}
```

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
        CGPoint location = [[touches anyObject] locationInView:self];

        CGFloat normalisedX = (location.x / RADIUS) - 1;
        CGFloat normalisedY = ((location.y / RADIUS) - 1) * -1;

        if (normalisedX > 1.0)
        {
           location.x = [self bounds].size.width;
           normalisedX = 1.0;
        }
        else if (normalisedX < -1.0)
        {
           location.x = 0.0;
           normalisedX = -1.0;
        }

        if (normalisedY > 1.0)
        {
           location.y = 0.0;
           normalisedY = 1.0;
        }
        else if (normalisedY < -1.0)
        {
           location.y = [self bounds].size.height;
           normalisedY = -1.0;
        }

        if (self.invertedYAxis)
        {
           normalisedY *= -1;
        }

        _xValue = normalisedX;
        _yValue = normalisedY;

        CGRect handleImageFrame = [_handleImageView frame];
        handleImageFrame.origin = CGPointMake(location.x - ([_handleImageView
bounds].size.width / 2),
                                              location.y - ([_handleImageView bounds].size.width /
2));
        [_handleImageView setFrame:handleImageFrame];

        if ([self.delegate respondsToSelector:@selector(analogueStickDidChangeValue:)])
        {
           [self.delegate analogueStickDidChangeValue:self];
        }

   _tocando = true;
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
        CGPoint location = [[touches anyObject] locationInView:self];

        CGFloat normalisedX = (location.x / RADIUS) - 1;
        CGFloat normalisedY = ((location.y / RADIUS) - 1) * -1;
   _tocando = true;
```

```
        if (normalisedX > 1.0)
        {
            location.x = [self bounds].size.width;
            normalisedX = 1.0;
        }
        else if (normalisedX < -1.0)
        {
            location.x = 0.0;
            normalisedX = -1.0;
        }

        if (normalisedY > 1.0)
        {
            location.y = 0.0;
            normalisedY = 1.0;
        }
        else if (normalisedY < -1.0)
        {
            location.y = [self bounds].size.height;
            normalisedY = -1.0;
        }

        if (self.invertedYAxis)
        {
            normalisedY *= -1;
        }

        _xValue = normalisedX;
        _yValue = normalisedY;

        CGRect handleImageFrame = [_handleImageView frame];
        handleImageFrame.origin = CGPointMake(location.x - ([_handleImageView
bounds].size.width / 2),
                                              location.y - ([_handleImageView bounds].size.width /
2));
        [_handleImageView setFrame:handleImageFrame];

        if ([self.delegate respondsToSelector:@selector(analogueStickDidChangeValue:)])
        {
            [self.delegate analogueStickDidChangeValue:self];
        }
}

- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
{
        _xValue = 0.0;
        _yValue = 0.0;
    _tocando = false;

        CGRect handleImageFrame = [_handleImageView frame];
        handleImageFrame.origin = CGPointMake(([self bounds].size.width - [_handleImageView
bounds].size.width) / 2,
                                              ([self bounds].size.height - [_handleImageView
bounds].size.height) / 2);
        [_handleImageView setFrame:handleImageFrame];

        if ([self.delegate respondsToSelector:@selector(analogueStickDidChangeValue:)])
        {
            [self.delegate analogueStickDidChangeValue:self];
        }
}
```

```
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
        _xValue = 0.0;
        _yValue = 0.0;
    _tocando = false;

        CGRect handleImageFrame = [_handleImageView frame];
        handleImageFrame.origin = CGPointMake(([self bounds].size.width - [_handleImageView
bounds].size.width) / 2,
                                              ([self bounds].size.height - [_handleImageView
bounds].size.height) / 2);
        [_handleImageView setFrame:handleImageFrame];

        if ([self.delegate respondsToSelector:@selector(analogueStickDidChangeValue:)])
        {
            [self.delegate analogueStickDidChangeValue:self];
        }
}

@end
```

11.2.2 Ayuda

- **AyudaViewController.h**

```
#import <UIKit/UIKit.h>

@interface AyudaViewController : UIViewController

@property (weak, nonatomic) IBOutlet UITextView *texto;

@end
```

- **AyudaViewController.m**

```
#import "AyudaViewController.h"

@interface AyudaViewController ()

@end

@implementation AyudaViewController

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];
        // Do any additional setup after loading the view.
    //_etiquetaAyuda.frame = CGRectMake(10, 10, 280, 280);
}

- (void) viewDidAppear:(BOOL)animated {
```

```
    //_etiquetaAyuda.frame = CGRectMake(10, 10, 280, 280);
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

11.2.3 Web

- **WebViewController.h**

```
#import <UIKit/UIKit.h>

@interface WebViewController : UIViewController

@property (weak, nonatomic) IBOutlet UIWebView *WebView;


@end
```

- **WebViewController.m**

```
#import "WebViewController.h"

@interface WebViewController ()

@end

@implementation WebViewController

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    NSURL *url = [NSURL URLWithString:@"http://www.marinaanaya.com/"];
    NSURLRequest *request = [NSURLRequest requestWithURL:url];
    [_WebView loadRequest:request];
    _WebView.scalesPageToFit = true;
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

11.2.4 Controles

- JoystickViewController.h

```
#import <UIKit/UIKit.h>
#import <CoreMotion/CoreMotion.h>
#import "JSAnalogueStick.h"
#import "AppDelegate.h"
#import "BLE.h"

#define maxX 0.55       // Rangos del acelerómetro
#define minX 0.25
#define maxY 0.3
#define minY -0.3

@interface JoystickViewController : UIViewController <BLEDelegate, JSAnalogueStickDelegate,
UIAccelerometerDelegate>

{
    BLE *bleShield;
    NSInteger dispositivos[5];
    NSInteger bateria;
    bool enable;        // Me permite controlar si he recibido mensaje de confirmacion de la plataforma
para poder enviar otro mensaje nuevo
    bool acelOn;        // Me permite seleccionar uno u otro método de control
    bool desconexion;   // Para no enviar mas resets si se esta desconectando
    bool parada;        // Para enviar solo un reset y permitir la desconexion cuando se sale de la app
y no por el boton
    bool disponible;    // Para saber si la placa esta siendo usada o no
    NSTimer *temporizador;  // Para controlar el tiempo máximo de búsqueda-conexión
}

@property (weak, nonatomic) IBOutlet UIButton *botonConectar;
@property (weak, nonatomic) IBOutlet UIButton *botonDesconectar;
@property (weak, nonatomic) IBOutlet UIButton *botonRob1;
@property (weak, nonatomic) IBOutlet UIButton *botonRob2;
@property (weak, nonatomic) IBOutlet UISegmentedControl *segmentedController;
@property (weak, nonatomic) IBOutlet UISlider *servo1;
@property (weak, nonatomic) IBOutlet UISlider *servo2;
@property (weak, nonatomic) IBOutlet UISwitch *led;
@property (weak, nonatomic) IBOutlet UIActivityIndicatorView *spinner;
@property (weak, nonatomic) IBOutlet UIProgressView *barraProgreso;
@property (weak, nonatomic) IBOutlet JSAnalogueStick *analogueStick;
@property (weak, nonatomic) IBOutlet UIImageView *imagenJoystick;
@property(readonly, nonatomic) CMMotionManager *acelerometro;

-(void) conexion;
-(void) desconexion;
-(void) connectionTimer:(NSTimer *) timer;
-(void) connectionTimer2:(NSTimer *) timer;
-(void) connectionTimer3:(NSTimer *) timer;
-(void) timeoutTimer:(NSTimer *) timer;
-(void) bateriaBaja;
-(void) actualizarJoystick;
-(void) cambioValorAcelerometro:(CMAcceleration)acceleration;
-(void) appGoingToSleep;
-(void) dealloc1;

@end
```

- **JoystickViewController.m**

```
#import "JoystickViewController.h"

@interface JoystickViewController ()

@end

@implementation JoystickViewController

@synthesize acelerometro;
@synthesize segmentedController;

#pragma mark - Inicialización

- (void)viewDidLoad
{
    [super viewDidLoad];

    bleShield = [[BLE alloc] init];                 // Inicializo el objeto Bluetooth
    [bleShield controlSetup];
    bleShield.delegate = self;

    for (int i=0; i<5; i++) dispositivos[i]=0;

    self.botonRob1.hidden = true;
    self.botonRob2.hidden = true;
    self.botonDesconectar.hidden = true;

    self.servo1.hidden = true;
    self.servo2.hidden = true;
    self.led.hidden = true;
    self.segmentedController.hidden = true;
    self.imagenJoystick.hidden = true;
    _analogueStick.hidden = true;

    [self.spinner startAnimating];
    self.spinner.hidden = true;
    self.barraProgreso.hidden = true;

    acelerometro = [[CMMotionManager alloc] init];  //Inicializo el acelerometro
    self.acelerometro.accelerometerUpdateInterval = .2;

    [self.acelerometro startAccelerometerUpdatesToQueue:[NSOperationQueue currentQueue]
withHandler:^(CMAccelerometerData  *accelerometerData, NSError *error)         //Asocio los
cambios en el acelerometro a la funcion cambioValorAcelerometro
     {
         [self cambioValorAcelerometro:accelerometerData.acceleration];
         if(error) NSLog(@"%@", error);
     }];
}

-(void)viewDidAppear:(BOOL)animated
{
    [super viewWillAppear: animated];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
}

#pragma mark - Acciones
```

```
- (IBAction)conectar:(id)sender {
    [bleShield.peripherals removeAllObjects];
    [bleShield findBLEPeripherals:1];
    [NSTimer scheduledTimerWithTimeInterval:(float)2.0 target:self
selector:@selector(connectionTimer:) userInfo:nil repeats:NO];
    temporizador = [NSTimer scheduledTimerWithTimeInterval:(float)20.0 target:self
selector:@selector(timeoutTimer:) userInfo:nil repeats:NO];
    for (int i=0; i<5; i++) dispositivos[i]=0;
    self.botonConectar.hidden = true;
    self.botonRob1.hidden = true;
    self.botonRob2.hidden = true;
    self.spinner.hidden = false;
    self.barraProgreso.hidden = false;
    self.barraProgreso.progress = 0.25;
}

- (IBAction)conectarRob1:(id)sender {
    [bleShield connectPeripheral:[bleShield.peripherals objectAtIndex:dispositivos[1]]];
    self.botonRob1.hidden = true;
    self.botonRob2.hidden = true;
    self.botonConectar.hidden = true;
    dispositivos[3]=4;
}

- (IBAction)conectarRob2:(id)sender {
    [bleShield connectPeripheral:[bleShield.peripherals objectAtIndex:dispositivos[2]]];
    self.botonRob1.hidden = true;
    self.botonRob2.hidden = true;
    self.botonConectar.hidden = true;
    dispositivos[4]=4;
}

- (IBAction)desconectar:(id)sender {
    [self desconexion];
}

- (IBAction)segmentedController:(id)sender {
    if(segmentedController.selectedSegmentIndex == 0) {
        _analogueStick.hidden = false;
        acelOn = false;
    }
    else if(segmentedController.selectedSegmentIndex == 1) {
        _analogueStick.hidden = true;
        acelOn = true;
    }
}

- (IBAction)servo1:(id)sender {
    if (enable) {
        UInt8 buf[] = {0x02, 0x00, 0x00};
        buf[1]=(UInt8)_servo1.value;
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        enable = false;
    }
}

- (IBAction)servo2:(id)sender {
    if (enable) {
        UInt8 buf[] = {0x03, 0x00, 0x00};
```

```
        buf[1]=(UInt8)_servo2.value;
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        enable = false;
    }
}

- (IBAction)led:(id)sender {
    if (enable) {
        UInt8 buf[] = {0x04, 0x00, 0x00};
        if (_led.isOn) buf[1] = 0x01;
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        enable = false;
    }
}

#pragma mark - Desconexión

-(void) desconexion
{
    enable = false;
    acelOn = false;
    desconexion = true;
    segmentedController.selectedSegmentIndex = 0;
    self.spinner.hidden = false;
    self.barraProgreso.progress = 0.25;
    self.barraProgreso.hidden = false;
    self.botonDesconectar.hidden = true;
    _analogueStick.hidden = true;
    self.servo1.hidden = true;
    self.servo2.hidden = true;
    self.led.hidden = true;
    self.segmentedController.hidden = true;
    self.imagenJoystick.hidden = true;

    [NSTimer scheduledTimerWithTimeInterval:(float)1.0 target:self
selector:@selector(connectionTimer2:) userInfo:nil repeats:NO];
}

-(void) connectionTimer2:(NSTimer *) timer
{
    self.barraProgreso.progress += 0.5;
    enable = false;
    UInt8 buf[] = {0x06, 0x03, 0x00};
    NSData *data = [[NSData alloc] initWithBytes:buf length:3];
    [bleShield write:data];
    [NSTimer scheduledTimerWithTimeInterval:(float)2.0 target:self
selector:@selector(connectionTimer3:) userInfo:nil repeats:NO];

    [self dealloc1];
}

-(void) connectionTimer3:(NSTimer *) timer
{
    [[bleShield CM] cancelPeripheralConnection:[bleShield activePeripheral]];
    self.barraProgreso.hidden = true;
    self.spinner.hidden = true;
    self.botonConectar.hidden = false;
    desconexion = false;
    enable = false;
```

```
    parada = false;
    for (int i=0; i<5; i++) dispositivos[i]=0;
}

-(void) bleCambioEstadoBluetooth
{
    if (dispositivos[3] == 5 || dispositivos[4] == 5) {
        self.botonDesconectar.hidden = true;
        _analogueStick.hidden = true;
        self.servo1.hidden = true;
        self.servo2.hidden = true;
        self.led.hidden = true;
        self.segmentedController.hidden = true;
        self.imagenJoystick.hidden = true;
        for (int i=0; i<5; i++) dispositivos[i]=0;
        enable = false;
    }
    self.botonConectar.hidden = false;
    self.spinner.hidden = true;
    self.barraProgreso.hidden = true;
}

-(void) timeoutTimer:(NSTimer *) timer
{
    NSLog(@"superado el tiempo muerto");
    self.botonConectar.hidden = false;
    self.spinner.hidden = true;
    self.barraProgreso.hidden = true;

    if (bleShield.activePeripheral) {          // Solo cancelamos la conexión si nos habíamos conectado
previamente
        [[bleShield CM] cancelPeripheralConnection:[bleShield activePeripheral]];
    }
}

#pragma mark - Conexión y comprobación

-(void) connectionTimer:(NSTimer *) timer
{

    if (bleShield.peripherals.count >0) dispositivos[0]=bleShield.peripherals.count;
    else {
        self.botonConectar.hidden = false;
        if (temporizador) {
            NSLog(@"Invalidamos el temporizador del timeout");
            [temporizador invalidate];
            temporizador = nil;
        }
    }

    [self conexion];
}

-(void) conexion
{
    NSLog(@"rutina conexion");
    if (dispositivos[0] > 0){
        dispositivos[0]--;
        [bleShield connectPeripheral:[bleShield.peripherals objectAtIndex:dispositivos[0]]];
        self.barraProgreso.progress += 0.25;
    }
```

```
    else {                                  // Se muestra el estado de los dispositivos, o los controles
        self.barraProgreso.hidden = true;       // en caso de haberme conectado
        self.spinner.hidden = true;
        if (dispositivos[3] == 2) {
            self.botonRob1.hidden = false;
            self.botonRob1.enabled = true;
        }
        else if (dispositivos[3] == 3) {
            self.botonRob1.hidden = false;
            self.botonRob1.enabled = false;
        }
        if (dispositivos[4] == 2) {
            self.botonRob2.hidden = false;
            self.botonRob2.enabled = true;
        }
        else if (dispositivos[4] == 3) {
            self.botonRob2.hidden = false;
            self.botonRob2.enabled = false;
        }
        self.botonConectar.hidden = false;
        if (dispositivos[3] == 5 || dispositivos[4] == 5) {
            enable = true;
            self.botonConectar.hidden = true;
            self.botonRob1.hidden = true;
            self.botonRob2.hidden = true;
            self.botonDesconectar.hidden = false;
            _analogueStick.hidden = false;
            self.servo1.hidden = false;
            self.servo2.hidden = false;
            self.led.hidden = false;
            self.segmentedController.hidden = false;
            self.imagenJoystick.hidden = false;
            self.servo1.value = 0;
            self.servo2.value = 0;
            self.led.on = false;
            [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(applicationDidEnterBackground:)
name:UIApplicationDidEnterBackgroundNotification object:nil];
        }
    }

}

-(void) bleConectado
{
    if (dispositivos[3] == 4 || dispositivos[4] == 4) {
        NSLog(@"Me conecto al dispositivo");
        UInt8 buf[] = {0x06, 0x02, 0x00};
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        return;
    }
    else {
        NSLog(@"envio dato de comprobacion");
        UInt8 buf[] = {0x06, 0x01, 0x00};
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        return;
    }
}
```

```
-(void) bleDesconectado
{
    NSLog(@"Desconectado");
    if (dispositivos[3] == 0 && dispositivos[4] == 0) {
        self.botonConectar.hidden = false;
        self.spinner.hidden = true;
        self.barraProgreso.hidden = true;
    }
    if (temporizador) {
        NSLog(@"Invalidamos el temporizador del timeout");
        [temporizador invalidate];
        temporizador = nil;
    }
}

-(void) bleNoPudoConectar
{
    self.botonConectar.hidden = false;
    self.spinner.hidden = true;
    self.barraProgreso.hidden = true;
    if (temporizador) {
        NSLog(@"Invalidamos el temporizador del timeout");
        [temporizador invalidate];
        temporizador = nil;
    }
}

-(void) bleNuevoDato:(unsigned char *)data length:(int)length   //cuando recibamos un dato nuevo
{
    if (temporizador) {
        NSLog(@"Invalidamos el temporizador del timeout");
        [temporizador invalidate];
        temporizador = nil;
    }
    NSLog(@"Recibimos dato de la plataforma %hhu,%hhu,%hhu",data[0],data[1],data[2]);
    if (data[0] == 0x06 && (dispositivos[3] != 5 || dispositivos[4] != 5)) {
        if (data[1] == 0x01) {                          // 0x01 para el robot 1
            if (dispositivos[0] == 1) {
                dispositivos[1] = 1;                    // Le digo al programa en que indice esta rob1
                dispositivos[2] = 0;                    // Le digo al programa en que indice esta rob2
            }

            if (data[2] == 0x01) dispositivos[3] = 2;          // 2.Disponible
            else if (data[2] == 0x02) dispositivos[3] = 3;     // 3.No disponible
            else if (data[2] == 0x03) {                     // 5.Utilizando   -   4 equivale a querer conectarse
                dispositivos[3] = 5;
                [self conexion];
                return;
            }
        }
        else if (data[1] == 0x02) {                     // 0x02 Para el robot 2
            if (dispositivos[0] == 1){
                dispositivos[1] = 0;                    // Le digo al programa en que indice esta rob1
                dispositivos[2] = 1;                    // Le digo al programa en que indice esta rob2
            }

            if (data[2] == 0x01) dispositivos[4] = 2;
            else if (data[2] == 0x02) dispositivos[4] = 3;
            else if (data[2] == 0x03) {
                dispositivos[4] = 5;
                [self conexion];
```

```
            return;
        }
    }
}
else if (data[0] == 0x07) {
    [self desconexion];
    return;
}
else if (data[0] == 0x08){
    bateria = data[1]*100+data[2];
    if (bateria <= 708) [self bateriaBaja];
    if (bateria > 708 && (dispositivos[3] == 5 || dispositivos[4] == 5)) enable = true; // habilitamos al
iPhone para enviar un dato

    return;
}
NSLog(@"dispositivos:
%ld,%ld,%ld,%ld,%ld",(long)dispositivos[0],(long)dispositivos[1],(long)dispositivos[2],(long)dispositi
vos[3],(long)dispositivos[4]);
[[bleShield CM] cancelPeripheralConnection:[bleShield activePeripheral]];
[self conexion];
}


-(NSString*)getUUIDString:(CFUUIDRef)ref {
NSString *str = [NSString stringWithFormat:@"%@",ref];
return [[NSString stringWithFormat:@"%@",str] substringWithRange:NSMakeRange(str.length -
36, 36)];
}

#pragma mark - Control de la batería

-(void) bateriaBaja
{
    segmentedController.selectedSegmentIndex = 0;
    self.botonDesconectar.hidden = true;
    _analogueStick.hidden = true;
    self.servo1.hidden = true;
    self.servo2.hidden = true;
    self.led.hidden = true;
    self.segmentedController.hidden = true;
    self.imagenJoystick.hidden = true;
    enable = false;
    desconexion = true;

    UInt8 buf[] = {0x06, 0x03, 0x00};
    NSData *data = [[NSData alloc] initWithBytes:buf length:3];
    [bleShield write:data];
    [NSTimer scheduledTimerWithTimeInterval:(float)1.0 target:self
selector:@selector(connectionTimer3:) userInfo:nil repeats:NO];
    NSLog(@"envio la desconexion");

    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"BATERIA BAJA" message:@"Por favor,
avise a la persona encargada de la instalación. Gracias" delegate:nil cancelButtonTitle:@"OK"
otherButtonTitles:nil];
    [alert show];
}

#pragma mark - JSAnalogueStickDelegate

- (void) actualizarJoystick
```

```
{
    if (enable){

    NSLog(@"enviando dato");

    UInt8 buf[] = {0x01, 0x00, 0x00};
    buf[1]=(self.analogueStick.yValue*50 + 50) + (self.analogueStick.xValue*45);
    buf[2]=(self.analogueStick.yValue*50 + 50) - (self.analogueStick.xValue*45);

        if (buf[1] < 0 || buf[1] > 200) buf[1]=0;
        if (buf[2] < 0 || buf[2] > 200) buf[2]=0;
        if (buf[1] > 100) buf[1]=100;
        if (buf[2] > 100) buf[2]=100;

    NSData *data = [[NSData alloc] initWithBytes:buf length:3];
    [bleShield write:data];
    parada = false;
    enable = false;
    }

    NSString *flecha;
    if (self.analogueStick.yValue < 0.2 & self.analogueStick.yValue > -0.2) {   // Mostramos las
flechas
        if (self.analogueStick.xValue > 0.2) flecha = @"rotad";
        else if (self.analogueStick.xValue < -0.2) flecha = @"rotai";
        else flecha = @"stop";
    }
    if (self.analogueStick.xValue < 0.2 & self.analogueStick.xValue > -0.2) {
        if (self.analogueStick.yValue < -0.2) flecha = @"abajo";
        else if (self.analogueStick.yValue > 0.2) flecha = @"arriba";
        else flecha = @"stop";
    }
    if (self.analogueStick.yValue > 0.2) {
        if (self.analogueStick.xValue < -0.2) flecha = @"arribai";
        else if (self.analogueStick.xValue > 0.2) flecha = @"arribad";
        else flecha = @"arriba";
    }
    if (self.analogueStick.yValue < -0.2) {
        if (self.analogueStick.xValue < -0.2) flecha = @"abajod";
        else if (self.analogueStick.xValue > 0.2) flecha = @"abajoi";
        else flecha = @"abajo";
    }

    UIImage *img = [UIImage imageWithContentsOfFile:[[NSBundle mainBundle]
pathForResource:flecha ofType:@"png"]];
    [_imagenJoystick setImage:img];        // Ponemos la imagen correspondiente al movimiento
}

- (void)analogueStickDidChangeValue:(JSAnalogueStick *)analogueStick
{
        [self actualizarJoystick];
}

#pragma mark - Control del Acelerometro

-(void)cambioValorAcelerometro:(CMAcceleration)acceleration
{
    if (_analogueStick.tocando == false && enable == true && acelOn == false && desconexion ==
false && parada == false) {
        parada = true;                          // Esta rutina se ejecuta con una alta frecuencia y aqui
comprobamos
```

```
        enable = false;                             // si se esta tocando el joystick o no, mientras estamos
conectados
        UInt8 buf[] = {0x05, 0x00, 0x00};
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        NSLog(@"ponemos motores a 0");
    }
    if (acelOn) {
        if (enable && !desconexion){
            NSLog(@"enviando dato");
            UInt8 buf[] = {0x01, 0x00, 0x00};
            buf[1]=(-(acceleration.x-0.25)*60 + 60) + (acceleration.y*45);
            buf[2]=(-(acceleration.x-0.25)*60 + 60) - (acceleration.y*45);

            if (buf[1] < 0 || buf[1] > 200) buf[1]=0;
            if (buf[2] < 0 || buf[2] > 200) buf[2]=0;
            if (buf[1] > 100) buf[1]=100;
            if (buf[2] > 100) buf[2]=100;

            NSData *data = [[NSData alloc] initWithBytes:buf length:3];
            [bleShield write:data];
            parada = false;
            enable = false;
        }
        NSString *flecha;
        if (acceleration.x < maxX & acceleration.x > minX) {    // Mostramos las flechas
            if (acceleration.y > maxY) flecha = @"rotad";
            else if (acceleration.y < minY) flecha = @"rotai";
            else flecha = @"stop";
        }
        if (acceleration.y < maxY & acceleration.y > minY) {
            if (acceleration.x > maxX) flecha = @"abajo";
            else if (acceleration.x < minX) flecha = @"arriba";
            else flecha = @"stop";
        }
        if (acceleration.x < minX) {
            if (acceleration.y < minY) flecha = @"arribai";
            else if (acceleration.y > maxY) flecha = @"arribad";
            else flecha = @"arriba";
        }
        if (acceleration.x > maxX) {
            if (acceleration.y < minY) flecha = @"abajod";
            else if (acceleration.y > maxY) flecha = @"abajoi";
            else flecha = @"abajo";
        }

        UIImage *img = [UIImage imageWithContentsOfFile:[[NSBundle mainBundle]
pathForResource:flecha ofType:@"png"]];
        [_imagenJoystick setImage:img];     // Ponemos la imagen correspondiente al movimiento
    }
}

#pragma mark - Cuando salimos de la aplicacion

- (void)applicationDidEnterBackground:(NSNotification *)notification
{
    [self appGoingToSleep];
    desconexion = true;
    enable = false;
}
```

```
-(void) appGoingToSleep
{
    self.botonDesconectar.hidden = true;
    _analogueStick.hidden = true;
    self.servo1.hidden = true;
    self.servo2.hidden = true;
    self.led.hidden = true;
    self.segmentedController.hidden = true;
    self.imagenJoystick.hidden = true;
    NSLog(@"appGoingToSleep");

    if (dispositivos[3] == 5 || dispositivos[4] == 5) {
        UInt8 buf[] = {0x06, 0x03, 0x00};
        NSData *data = [[NSData alloc] initWithBytes:buf length:3];
        [bleShield write:data];
        [NSTimer scheduledTimerWithTimeInterval:(float)1.0 target:self
selector:@selector(connectionTimer3:) userInfo:nil repeats:NO];
        NSLog(@"envio la desconexion");
    }
    [self dealloc1];
}

-(void)dealloc1
{
    [[NSNotificationCenter defaultCenter] removeObserver:self
name:UIApplicationDidEnterBackgroundNotification object:nil];
}

@end
```

11.3 Código del Arduino

```
/*Proyecto fin de carrera de Alejandro Ibañez. PFCBluetooth*/

#include <ble_mini.h>
#include <Servo.h>
#include <Ultrasonic.h>

#define SERVO1_PIN      10
#define SERVO2_PIN      11
#define LED_PIN         13   //Dado que no es una PWM

#define MI2_PIN         6
#define MI1_PIN         5
#define MD1_PIN         3
#define MD2_PIN         9
#define ENABLE_PIN      2

#define DDT_PIN         7    //Ultrasonido Delante Derecho TRIG_PIN
#define DDE_PIN         A2   //Ultrasonido Delante Derecho ECHO_PIN 18
#define DIT_PIN         8    //Ultrasonido Delante Izquierdo TRIG_PIN
#define DIE_PIN         A1   //Ultrasonido Delante Izquierdo ECHO_PIN 15
#define ADT_PIN         13   //Ultrasonido Atras Derecho TRIG_PIN
#define ADE_PIN         A4   //Ultrasonido Atras Derecho ECHO_PIN 16
#define AIT_PIN         4    //Ultrasonido Atras Izquierdo TRIG_PIN
#define AIE_PIN         A3   //Ultrasonido Atras Izquierdo ECHO_PIN 17

#define BAT_PIN         A7   //Pin correspondiente a la bateria, para medir su tension
```

```
Servo     Servo1,Servo2;
int       M1,M2;
unsigned long     previousMillis,currentMillis,tiempo;
int       usando=0,parada=0,bateria=0,cambio=0;

Ultrasonic ultraAtrDcha(ADT_PIN,ADE_PIN);   // (Trig PIN,Echo PIN)
Ultrasonic ultraAtrIzq(AIT_PIN,AIE_PIN);
Ultrasonic ultraDelDcha(DDT_PIN,DDE_PIN);
Ultrasonic ultraDelIzq(DIT_PIN,DIE_PIN);

void ultrasonidos(int d1,int d2) {    //Medimos la distancia a un objeto cercano en la direccion
  long distancia;

  if (cambio == 0) cambio=1;          //Cambiamos de ultrasonidos a comprobar
  else cambio=0;

  if(d1>=0 && d2>=0){
    if(cambio == 0) distancia=ultraDelDcha.Ranging(CM);  // Obtenemos la distancia al objeto
    else distancia=ultraDelIzq.Ranging(CM);
  }
  else if (d1<0 && d2<0){
    if(cambio == 0) distancia=ultraAtrDcha.Ranging(CM);     //NOTA, dado al cambio de
movimiento, hacia atras
    else distancia=ultraAtrIzq.Ranging(CM);             //estan cambiados IZQ con DCHA
  }

  if (distancia < 18) parada=2;     // Si estamos a menos de 4cm detenemos en robot
  else {
    if (parada == 2) parada=1;      // Comprobamos dos iteraciones antes de permitir el movimiento
    else if (parada == 1) parada=0;
  }
}

int medir() {                       // Medimos el nivel de tension de la bateria
  int i = analogRead(BAT_PIN);
  if (i < 700) return 1;
  else return 0;
}

void bateriaBaja() {
  for(int pos = 0; pos < 180; pos += 5) {
    Servo1.write(pos);
    delay(15);
  }
  for(int pos = 180; pos >= 1; pos -= 5) {
    Servo1.write(pos);
    delay(15);
  }
}

void enviar(unsigned char a, unsigned char b, unsigned char c) {    // Enviamos un dato
  BLEMini_write(a);
  BLEMini_write(b);
  BLEMini_write(c);
```

```
}

void listo () { // Envio el dato de enable de la plataforma que corresponda
  uint16_t value = analogRead(BAT_PIN);              // Leemos y enviamos el nivel de tension de la bateria
  enviar(0x08,value/100,value-(value/100)*100);    // bat=1234 b1=12,b2=34
}

void desconexion() {
  analogWrite(MI1_PIN,0);
  analogWrite(MI2_PIN,0);
  analogWrite(MD1_PIN,0);
  analogWrite(MD2_PIN,0);
  digitalWrite(ENABLE_PIN,LOW);
  digitalWrite(LED_PIN,LOW);
  Servo1.write(15);
  Servo1.write(15);
  usando=0;
}

void setup() {
  BLEMini_begin(57600);
  tiempo = 60000;              // Despues de 60s inactivo se desconecta del usuario actual. Tambien sirve
                        // para resetear la placa si se desconecta accidentalmente de la plataforma
  Servo1.attach(SERVO1_PIN);
  Servo2.attach(SERVO2_PIN);

  digitalWrite(ENABLE_PIN,HIGH);
}

void loop() {

  byte data0,data1,data2;

  while ( BLEMini_available() == 3 )
  {
    // leemos los tres bytes recibidos
    data0 = BLEMini_read();
    data1 = BLEMini_read();
    data2 = BLEMini_read();

    if (data0 == 0x01) // Comando para regular la velocidad
    {
      M1=data2*4-200;
      M2=data1*4-200;
      ultrasonidos(M1,M2);         // Comprobamos que no tenemos ningun obstaculo en la direccion actual
      previousMillis = millis();  // Lo ponemos al ppio de cada instruccion para que no se resetee
                            // tambien al intentar conectar
      if (parada == 0 && bateria == 0){
        if (M1 >= 0){
          analogWrite(MD1_PIN,M1);
          analogWrite(MD2_PIN,0);
        }
```

```
    else {
      digitalWrite(MD2_PIN,HIGH);
      analogWrite(MD1_PIN,M1+255);
    }
    if (M2 >= 0){
      analogWrite(MI1_PIN,M2);
      analogWrite(MI2_PIN,0);
    }
    else {
      analogWrite(MI2_PIN,abs(M2));
      analogWrite(MI1_PIN,0);
    }
  }
  else{
    analogWrite(MD1_PIN,0);
    analogWrite(MD2_PIN,0);
    analogWrite(MI1_PIN,0);
    analogWrite(MI2_PIN,0);
  }
  listo();                  // Con esto permitimos volver a recibir un dato
}
else if (data0 == 0x02)     // Comando para modificar Servo1
{
  previousMillis = millis();
  Servo1.write(data1);
  listo();
}
else if (data0 == 0x03)     // Comando para modificar Servo2
{
  previousMillis = millis();
  Servo2.write(data1);
  listo();
}
else if (data0 == 0x04)  // Comando para modificar el LED
{
  previousMillis = millis();
  if (data1 == 0x01) digitalWrite(ENABLE_PIN,HIGH);
  else digitalWrite(ENABLE_PIN,LOW);
  listo();
}
else if (data0 == 0x05)  // Comando para poner los motores a 0, cuando suelto analogstick
{
  analogWrite(MI1_PIN,0);
  analogWrite(MI2_PIN,0);
  analogWrite(MD1_PIN,0);
  analogWrite(MD2_PIN,0);
  listo();
}
else if (data0 == 0x06) // Comando para comprobar si la plataforma esta en uso
{
  if (data1 == 0x01){
    if (usando == 0) enviar(0x06,0x01,0x01);
    else enviar(0x06,0x01,0x02);
  }
  else if (data1 == 0x02){
```

```
      if (usando == 0){
        enviar(0x06,0x01,0x03);//cambiar (0x06, 0x0[1 u 2], 0x02) en funcion de ROB1 o ROB2
        digitalWrite(ENABLE_PIN,HIGH);
        usando=1;
        previousMillis = millis();
       }
       else {
        enviar(0x06,0x01,0x02);//cambiar (0x06, 0x0[1 u 2], 0x02) en funcion de ROB1 o ROB2
       }
      }
      else if (data1 == 0x03){
       usando = 0;
       desconexion();
      }
    }
  }
  if (usando == 1) {
    currentMillis = millis();
    if (currentMillis - previousMillis > tiempo) {
      enviar(0x07,0x00,0x00);
      desconexion();
    }
  }
  uint16_t value = analogRead(BAT_PIN);
  if (value <= 708)  bateriaBaja();
}
```